WANG

VS

Linker Reference
Release 7 Series

VS

Linker Reference
Release 7 Series

1st Edition — October 1988

715-1145

WANG

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVENUE, LOWELL, MA 01851 TEL. (508) 459-5000, TELEX 172108

Disclaimer of Warranties and Limitation of Liabilities

The staff of Wang Laboratories, Inc., has taken due care in preparing this manual. However, nothing contained herein modifies or alters in any way the standard terms and conditions of the Wang purchase, lease, or license agreement by which the product was acquired, nor increases in any way Wang's liability to the customer. In no event shall Wang or its subsidiaries be liable for incidental or consequential damages in connection with or arising from the use of the product, the accompanying manual, or any related materials.

Software Notice

All Wang Program Products (software) are licensed to customers in accordance with the terms and conditions of the Wang Standard Software License. No title or ownership of Wang software is transferred, and any use of the software beyond the terms of the aforesaid license, without the written authorization of Wang, is prohibited.

Warning

This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instructions manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A computing device, pursuant to Subpart J of Part 15 of FCC rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference, in which case the user, at his own expense, will be required to take whatever measures may be required to correct the interference.

CONTENTS

CONTENTS (continued)

CONTENTS (continued)

FIGURES

TABLES

HOW TO USE THIS MANUAL

INTENDED AUDIENCE

This manual is written for VS programmers who require the use of a linker. Familiarity with the VS Operating System is assumed.

STRUCTURE

This manual consists of 5 chapters. In addition, there are five appendixes, the last of which is a glossary. The material in each chapter and appendix is organized in the following way:

- Chapter 1 provides an overview of the VS LINKER. It defines Linker terms and describes the Linker user interface, Linker features and functions, and the Linking process. It also compares the Linker with the previous Linker and describes its operating environment.

- Chapter 2 describes the components of the link. Included is a full description of a new feature: shared subroutine libraries.

- Chapter 3 explains Linker functions and describes how the Linker can be run interactively or by a VS Procedure.

- Chapter 4 describes the entire linking process in detail.

- Chapter 5 provides all the GETPARM information that is required for writing VS Procedures for the Linker.

- Appendix A shows two link maps, one for a program link and one for the linking of a shared subroutine library (SSL).

- Appendix B explains the default selection process that the Linker uses if it finds sections with the same name.

- Appendix C describes the significance of base-dependent code and why it must be removed from a shared subroutine library (SSL).
- Appendix D lists and describes the Linker error messages.
- Appendix E provides a list of Linker terms and their definitions.

RELATED DOCUMENTATION

For further information on related topics, consult the following documents:

- *VS Operating System Services Reference* (715-0423A)
- *VS System Administrator's Reference* (715-0420)
- *VS Procedure Language Reference* (800-1205-06)
- *VS System User's Introduction* (715-0417A)

CHAPTER 1
INTRODUCTION TO THE VS LINKER

1.1 INTRODUCTION

The Wang VS LINKER (or simply, Linker) assembles separately compiled or assembled program modules into a single executable program. In addition, the Linker creates and maintains subroutine libraries. The Linker enables you to perform the following tasks:

- Link together specified program modules
- Replace linked program modules with other specified modules
- Create and maintain subroutine libraries of frequently used program modules
- Link subroutines from subroutine libraries as needed by the output program file being created
- Access program modules written by other programmers
- Access program modules written in various programming languages

Programs that call external subroutines must be linked, since the subroutines are not a physical part of the program. When all specified modules are linked, you have a single linked program file that consists of the main program file, specified input files, and called program modules. Figure 1-1 shows the creation of a linked program file.

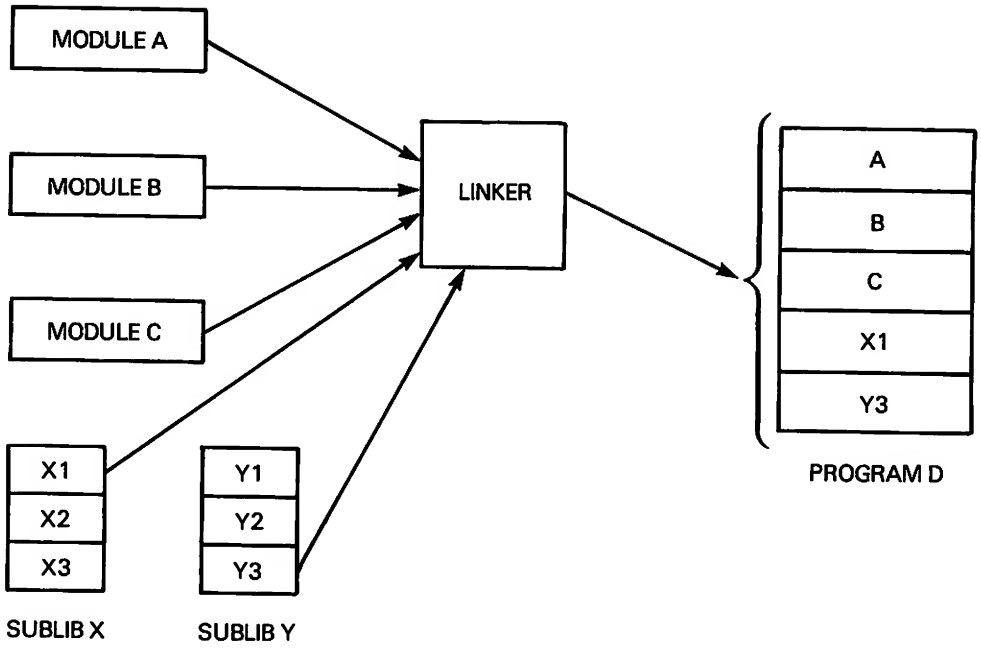

Figure 1-1. Creation of a Linked Program File

When you recompile one or more object modules of a linked program file, you can use the Linker to insert them into the file, replacing existing modules with the same name. You do not need to relink all the modules individually. You can perform this task by selectively choosing from the duplicate module names.

Figure 1-2 shows the replacement of an existing module in a linked program file.

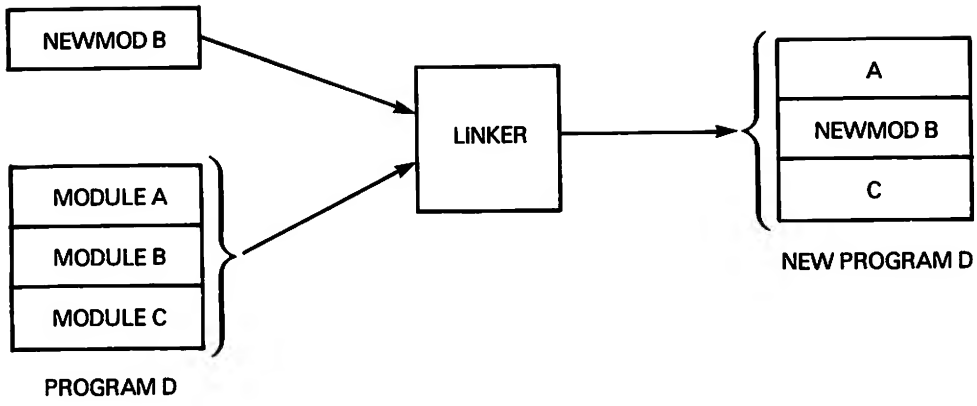

Figure 1-2. Replacement of an Existing Module in a Linked Program File

Note: *In cases where confusion might arise, the current version of the VS LINKER is referred to as the new Linker, and the previous version as the previous Linker. Otherwise, the current version is referred to as the Linker.*

The new Linker incorporates a more efficient method for organizing and linking subroutines than the previous Linker. In the past, there was one type of subroutine file that was added to the object file during the link. The entire file had to be linked, even if the program needed only certain subroutines contained in the file.

The new Linker links only the required subroutines of a subroutine file at linktime. It also utilizes a new type of subroutine library called a shared subroutine library, which is accessed at runtime rather than linktime, and can be shared by other programs.

1.2 LINKER TERMS

To effectively use the functions available with the Linker, you must be familiar with certain Linker terms. This section defines the key terms that all users must understand to perform the operations described in later chapters.

1.2.1 Program File

A program file is a consecutive file that contains 1024-byte records. It is distinguished from other types of files by having the program file flag set in its file descriptor record. The contents of a program file are restricted to a single object code format (refer to Section 1.2.2).

1.2.2 Object Code Format

An object code format is a set of rules that apply to the collection of blocks of information that make up a program file. Each block represents data required by a VS system component to accomplish any of the following tasks:

- Load the program into memory
- Perform symbolic debugging when the program is run
- Combine the program file with other program files

There are currently two object code formats described for the VS; Version 0 which applies to operating systems that precede Release 7.10, and Version 1 which applies to Release 7.10 and all subsequent operating systems. For more information on object code formats, refer to the *VS Operating System Services Reference*.

1.2.3 Program Section, Module, or Subroutine

A program section (also referred to as a module or subroutine), is a program unit that can be compiled or assembled independently of other units. Examples of program sections include PL/I modules, FORTRAN 77 labelled common areas, and static and code sections in Assembly language.

There are two types of program sections: static sections, which contain the program variables, and code sections, which contain the program statements that act on the variables.

1.2.4 Entry Point

An entry point is the point that initiates the processing of the linked program or module; that is, the point at which the execution of the program or module begins. An entry point name is the name of a symbol (refer to Section 1.2.5) that references an entry point.

1.2.5 Symbol

A symbol is either a program section name, an entry point name defined in the section, or an external reference (a reference to an entity located in a section outside of the current section). The value of a symbol is the entry point address of the entity that the symbol refers to. A symbol is defined (or resolved) if its value can be determined; otherwise, it is undefined (or unresolved).

Symbols are classified as either shared subroutine library symbols, or user-specified symbols. Shared subroutine library symbols are symbols that are defined in subroutines that can be shared between tasks at runtime. Examples of shared subroutine library symbols include symbols that refer to the routines in a shared subroutine library (refer to Section 1.2.8), DMS routines, and system services routines. Any symbols that are not shared subroutine library symbols are referred to as user-specified symbols.

1.2.6 Symbolic Reference

A symbolic reference is the use of a particular symbol at a location other than at the exact location where it is defined. For example, if Symbol A in Routine 1 is defined elsewhere in that routine, or in some other routine, then Symbol A is classified as a symbolic reference.

An internal symbolic reference refers to a symbol defined in the same section as the symbolic reference; an external symbolic reference refers to a symbol defined in a section other than the one in which it appears.

A *code symbolic reference* originates in a code section, while a *static symbolic reference* originates in a static section.

1.2.7 Static Subroutine Library

A static subroutine library contains subroutines that are selectively linked as needed at linktime, and cannot be shared by other programs at runtime. A link at linktime is referred to as a *static link*. The subroutines contained in a static subroutine library are called static subroutines.

The Linker supports a new type of static subroutine library that is contained in a single VS file. From this file, the Linker selects subroutines needed to resolve undefined symbols.

The previous type of static subroutine library is also supported for upward compatibility. This type consists of a single VS library, each file containing one or more subroutines. The Linker links one of the files only if its file name matches an undefined symbol.

1.2.8 Shared Subroutine Library

A shared subroutine library is a single VS file, which contains subroutines that are selectively accessed at runtime as opposed to linktime. Subroutines are loaded into memory as required, when programs that reference them are loaded. A link at runtime is referred to as a *dynamic link*.

For shared subroutines referred to in a program, the Linker inserts pointers to the subroutines referred to instead of incorporating the subroutines into the object code. Then, at runtime, the subroutines are accessed through the pointers. Subroutines in a shared subroutine library can be shared by multiple programs that are running concurrently.

1.2.9 Alias

An alias is a name of up to 40 characters assigned to each shared subroutine library by the system security administrator. This assignment is made using the SSL utility, which is described in the *VS System Administrator's Reference*.

The Linker uses aliases to refer to and verify shared subroutine libraries at linktime. At loadtime, when a program section refers to a module contained in a shared subroutine library, the Loader uses the alias of the shared subroutine library to access the subroutine and load it into memory prior to program execution.

1.2.10 Resolution

Resolution is the process of making an undefined (unresolved) symbol defined (resolved). *Static resolution* is performed by the Linker at linktime to resolve user-specified symbols only. *Dynamic resolution* is performed at runtime by the Run processor to resolve shared subroutine library symbols.

Individual program files, which are input to the Linker, generally contain external symbolic references; that is, references to symbols not defined in the input file. The Linker attempts to resolve these references by first searching for the symbols in the specified input files, then in static subroutine libraries, and finally in shared subroutine libraries. Note that the Linker searches shared subroutine libraries only if one or more aliases are supplied in the input file.

1.3 LINKER FEATURES

A summary of the significant Linker features follows:

Program File Linking -- You can select the program files to be linked. The Linker statically resolves any undefined user-specified symbols, or dynamically resolves any undefined shared subroutine library symbols.

Three Operational Modes -- You can run the Linker interactively in the foreground, by procedure in the foreground, or by procedure in the background.

Improved Static Subroutine Library Mechanism -- The Linker incorporates an improved static subroutine library mechanism that is not dependent on file names, and uses only required subroutines of subroutine library files.

Shared Subroutine Libraries -- The Linker enables you to create shared subroutine libraries and to utilize them in user programs.

Debugging -- The Linker fully supports the Debugger.

Help Facilities -- The Linker provides interactive program assistance in two forms: an on-line Help text facility and a displayable link map in progress.

Link Map -- The Linker enables you to create and print a link map: a comprehensive listing of the completed link.

Object Format Translation -- The Linker can be used to translate a Version 0 object format program file to a program file in Version 1 object format.

Optional Output Format -- The Linker enables you to specify an optional output format (Format 0) that is compatible with Release 6 of the operating system. The Release 7 operating system executes both Format 0 and Format 1 object code, but Release 6 and earlier operating systems recognize only Format 0: they cannot load and execute Format 1 object code.

1.4 LINKER FUNCTIONS

The Linker executes the following functions in a typical linking session:

- Accepts the input files, and selects from each file the code and static sections that are to be linked
- Resolves any undefined symbols, using user-specified input files, static subroutine libraries, and shared subroutine libraries
- Organizes the selected code and static sections into two separate sections, each of which will occupy a contiguous virtual memory location when loaded
- Optionally removes symbolic debug information, thereby reducing the size of the program, but preventing the use of symbolic debugging features
- Creates a list of symbols that were referenced in the link, and a list of corresponding aliases for shared subroutine library symbols
- Organizes the output file in Format 1 object code format (or optionally, Format 0), including any optionally specified information
- Optionally creates a link map print file
- Optionally reorders the linked sections in the output
- Optionally designates or modifies alias names for symbols referenced in the linked output
- Optionally displays a list of subroutine library entry point names for a static or shared subroutine library output file, and provides the option to select entry point names to be included in the output file

1.5 LINKER HELP FACILITIES

The Linker incorporates the following Help facilities to provide users with assistance if needed during the linking process. Since these features require the use of the workstation screen, the Linker can operate in Help mode only when it is executing in the foreground.

Help text -- You can invoke the VS INFO utility, an on-line information and help facility, to display user instructional Help text.

Link Map -- You can invoke the Display utility to examine portions of the link map in various stages of its creation.

1.5.1 Using Help Text

Help text is available from all Linker screens. It displays on-line information about the Linker screen from which it is referenced. Help text also provides conceptual information on the Linker, a glossary of terms, and error message definitions.

When you press PF13, the Linker accesses the VS INFO utility, which displays the first page of Help text for that screen. This contains a brief description of the purpose of the screen and several numeric references for more specific information.

You access specific information for the referenced screen, or for other portions of the Help text, by positioning the cursor on one of the reference numbers and pressing ENTER. The VS INFO utility automatically takes you to the portion of the Help text with that reference number. You can continue searching through the Help text by positioning the cursor on a reference number and pressing ENTER. Figure 1-3 shows a sample Linker Help text screen.

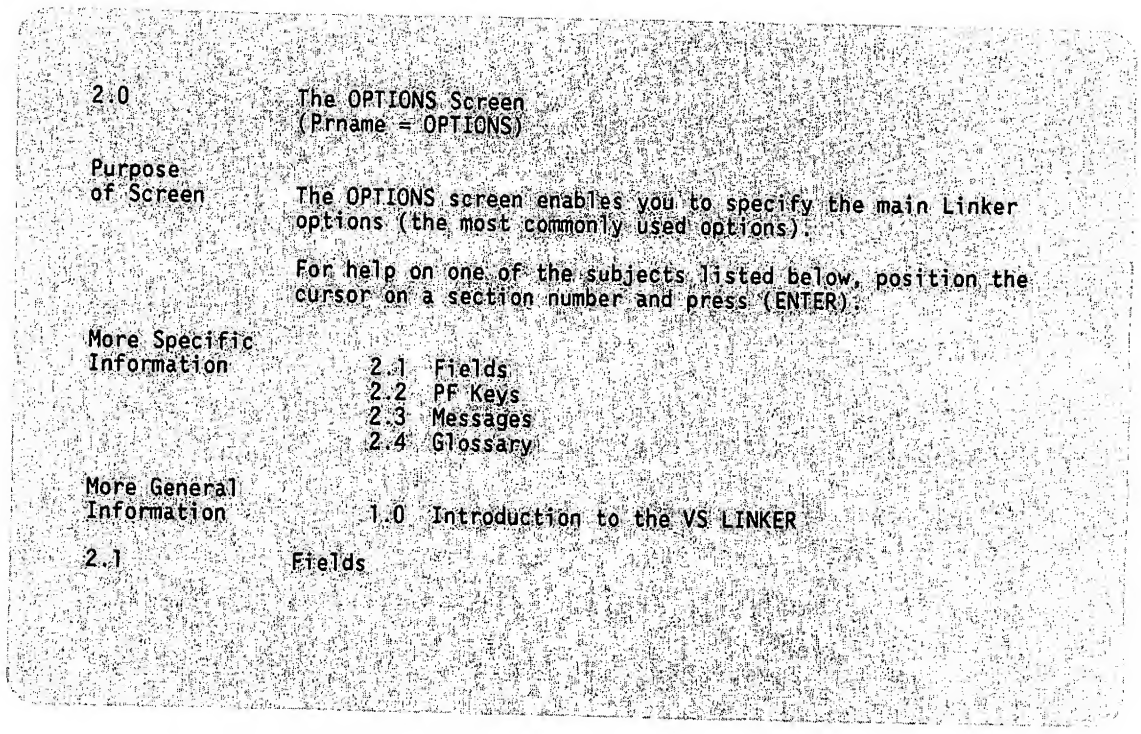

Figure 1-3. Sample Linker Help Text Screen

You can scroll through the Help text by using the following PF keys:

PF Key	Description
2	First -- Positions you at the beginning of the Help text for the Linker, i.e., Help Text 1.0.
3	Last -- Positions you at the end of the Help text for the Linker.
4	Prev -- Positions you at the previous screen of the Help text.
5	Next -- Positions you at the next screen of the Help text.
6	Down -- Positions you one line lower in the Help text.
7	Up -- Positions you one line higher in the Help text.

When you finish using the Help text, press PF16 to return to the Linker screen from which you invoked the Help text.

1.5.2 Using the Link Map

The link map displays extensive information about various results of the link, such as the input log, input files, linked code sections, etc. You can examine the contents of this map from most Linker screens, in various stages of its creation. This is done by pressing PF14. For detailed information on the link map, refer to Section 2.4.

1.6 THE LINKER USER INTERFACE

The Linker is menu-driven and guides you with clear screen explanations, prompts, and messages. You can create VS procedures that access and operate on most Linker screens. For information on creating VS procedures using Linker prnames and keywords, refer to Section 3.2.2 and Chapter 5.

1.6.1 Linker Screen Organization

Each Linker screen consists of the following sections:

- Screen title
- Instruction text
- List or attribute display
- PF keys

Figure 1-4 shows the FILELIST screen and the OUTPUT screen as sample Linker screens.

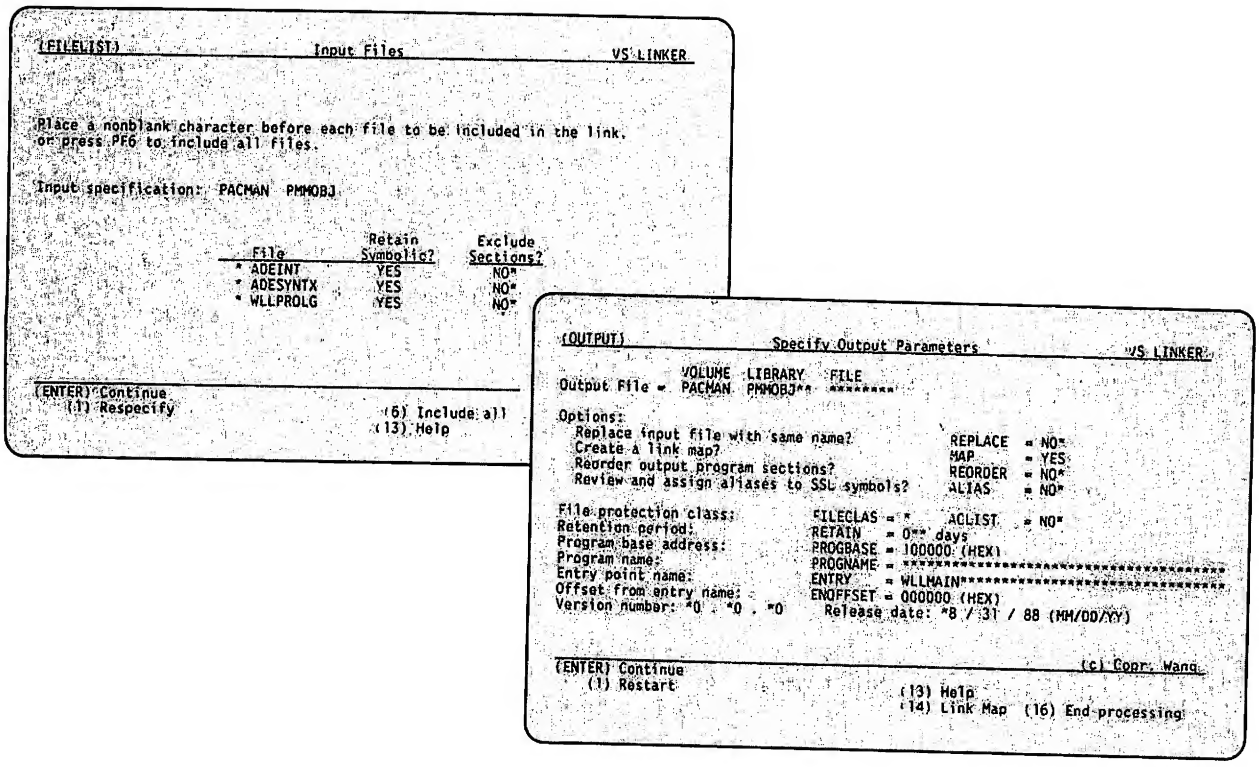

Figure 1-4. Sample Linker Screens

Most Linker screens have both an abbreviated screen title and a full screen title. The abbreviated screen title, which appears in the left hand corner of Line 1, is used in this manual to refer to the screen. It also serves as the prname for the screen if the screen can be used in a procedure. The full screen title, which appears centered on Line 1, indicates the service that the screen provides.

The instruction text briefly explains how to use the screen options. For example, the FILELIST screen instruction text tells you how to choose files to be included in the link. The List or Attribute Display section of any Linker screen contains either of the following entities:

- A list of items for selection
- The attributes for a specific item that you may wish to specify or manage

For example, the FILELIST screen lists files for you to select. The OUTPUT screen displays the attributes of the output file; some require entry of information; others require examination (those with supplied defaults).

The PF keys enable you to select functions listed on the screen.

1.7 OVERVIEW OF THE LINKING PROCESS

This section provides a general overview of the linking process. Each part refers to one or more Linker screens that are described in detail in later chapters.

The linking process consists of the following steps:

- Initiating the Linker
- Specifying the options to be used in the link
- Specifying the input file(s) to be used in the link
- Specifying static subroutine libraries, if needed, to resolve symbols
- Specifying more input files, or aliases for shared subroutine libraries, if needed, to resolve undefined symbols
- Including additional input information requested by the Linker, as a result of the options you chose
- Naming and describing the output file that will contain the linked program
- Including additional output information requested by the Linker, as a result of the output options you chose
- Building the output file

Figure 1-5 shows the process of creating a linked program.

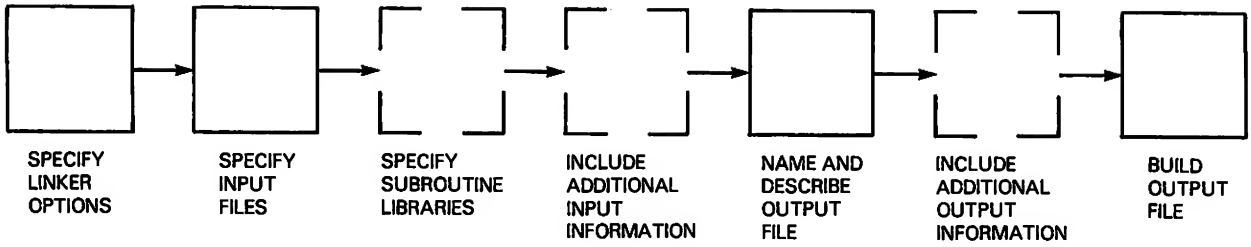

Figure 1-5. Creation of a Linked Program

1.7.1 Initiating the Linker

The Linker can be run by the Command Processor or a VS procedure. To access the Linker from the Command Processor menu, press PF1. The system displays the Run screen, which prompts you to enter the file, library, and volume names of the Linker. Since the Linker is stored in the system library on the system volume, you need only to enter its file name (LINKER) and press ENTER. The system then displays the initial Linker screen.

You can run the Linker by a procedure in either the foreground or background by using the Linker prnames and keywords in VS Procedure language programs. For more information, refer to Section 3.2.2. For detailed information on the VS Procedure language, refer to the *VS Procedure Language Reference*.

1.7.2 Specifying Linker Options

To begin creating a linked program, you first specify the options to be incorporated in the link. When initiated, the Linker displays a list of the most common Linker options from which you choose those appropriate for the desired link. Additional Linker options can be specified by pressing the PFkey for more options.

1.7.3 Specifying Input Program Files

After you have selected the Linker options, you next specify the input program files (input files) to be included in the link. You can enter one file at a time or indicate an entire library of files. If you specify a library, the Linker displays the contents of the library on a subsequent screen, from which you can select files.

To resolve external references in each input file, the Linker first searches all other input files. If undefined symbols exist following the input file search, the Linker enables you to specify static subroutine libraries, more input files, or shared subroutine libraries to resolve them.

1.7.4 Specifying Static Subroutine Libraries

If the Linker is unable to resolve undefined symbols, it allows you to specify one or more static subroutine libraries as part of the input. The Linker uses the appropriate subroutines from the static subroutine libraries that you specify to resolve undefined symbols.

1.7.5 Specifying More Input Files or Shared Subroutine Libraries

If the Linker is not able to resolve all undefined symbols by checking the static subroutine libraries specified, it notifies you that unresolved symbols still exist, and provides four options. You can build the output file with the undefined symbols, add more input files to resolve the undefined symbols, add to the list of aliases for shared subroutine libraries to resolve the undefined symbols, or display a list of undefined symbols and corresponding SSL symbols (if any were previously assigned), and add to or modify this list.

If you specify any option but the first, the Linker processes the information and informs you if any undefined symbols still exist. It then allows you to repeat any of the last three options until all the undefined symbols are resolved, or until you press ENTER to proceed to create the output file with undefined symbols.

1.7.6 Specifying Additional Input Information

The Linker may require additional information depending on the chosen Linker options. For example, if you choose the option to resolve conflicting section names, the Linker displays a screen that prompts you to correct any existing section name conflicts. After you have satisfied the requests for additional input information, the Linker continues processing.

1.7.7 Specifying the Output Program File

The next step in the linking process is to name and describe the output program file (output file). The Linker requests information on the output file name, location, options, and attributes.

1.7.8 Providing Additional Output Information

The Linker may require additional output information depending on the input options chosen or output file options specified. For example, if you choose the link map option, the Linker displays a screen that prompts you to specify the link map file and to select the portions of the link map to be printed. After you have satisfied the requests for additional information, you can continue with the link.

1.7.9 Building the Output File

When the Linker has shown all screens according to the Linker input and output options chosen, it proceeds to build the output file. After the Linker finishes building the output file, it returns control to the calling program.

1.8 COMPARISON TO THE PREVIOUS LINKER

The new Linker expands on the capabilities of the previous Linker and provides new functions not available with the previous Linker. The following list summarizes the major differences:

Note: *The term "fully qualified" means that an entity's volume, library, and file names have been specified. The term "partially qualified" means that the entity's volume and library names have been specified, but not the file name.*

- The new Linker accepts program files of both Version 0 and Version 1 object code; the previous Linker accepts only program files of Version 0 object code format.

- The new Linker produces an output file in either Version 0 or Version 1 object code format; the previous Linker produces an output file in Version 0 object code format.

- The new Linker accepts input files, informs you of unresolved external references, and then permits you to specify more input files, static subroutine libraries, or shared subroutine libraries to resolve them. With the previous Linker, you must specify static subroutine libraries before entering the input files. In addition, the previous Linker does not incorporate shared subroutine libraries.

- The link map for the new Linker contains more extensive information.

- The new Linker enables you to examine the sections of the link map during Linker execution.

- For multiple occurrences of a section name, the new Linker provides default selection rules, plus an override mode for selecting the appropriate section; the previous Linker provides only the default selection rules.

- The new Linker enables you to exclude any number of sections within an input file from the linking process.

- The new Linker provides on-line Help text through the VS INFO facility.

- The new Linker accepts both fully and partially qualified input file names; the previous Linker accepts only fully qualified input file names.

- The new Linker accepts both fully and partially qualified subroutine library names; the previous Linker accepts only fully qualified subroutine library names.

- To resolve an undefined symbol through a static subroutine library, the new Linker uses the matching entry point name in the library, and does not depend on the file name of the library to do the matching. The previous Linker resolves an undefined symbol through a static subroutine library by finding a matching file name in the library.

- The new Linker selects only those sections needed to satisfy references; the previous Linker must include all or none of a subroutine library file.

1.9 OPERATING ENVIRONMENT

The Linker runs on Operating System Release 7.10 and all subsequent releases of the VS Operating System.

CHAPTER 2
COMPONENTS OF THE LINK

2.1 INTRODUCTION

A linked program can consist of the following components:

- One or more input files (required)
- Symbolic debugging information associated with individual input files (optional)
- Static subroutines from one or more static subroutine libraries (optional)
- References to shared subroutines from one or more shared subroutine libraries (optional)
- A link map (optional)

This chapter defines each component, and describes the purpose, function, and management of the sections or objects within it. This chapter also describes the output file: the storage area for all linked components.

2.2 INPUT FILES

An input file to the Linker is called a program or object file. A program file has the following characteristics:

- It is the output of a compiler, assembler, or Linker.
- It is a consecutive file with 1024-byte records consisting of executable code, data, and other information.
- It is organized into one or more program sections.
- It is distinguished from other types of files by having the program file flag set in its File Descriptor Record.

A linked program consists of a collection of separately compiled or assembled program files combined together into one executable output file.

In the process of linking a program, you first select the options for the link. Then, you specify the input files (program files) to be used. The Linker accepts input files of either Version 0 or Version 1 object code format. Input files are specified to be included completely in the linking process, unless you exclude selected sections by using the EXCLUDE option (refer to Section 4.3.4).

You can specify input files singly, or enter the name of a library. If you specify a library, the Linker displays the object files contained in that library on a separate screen. From that screen, you can select any number of input files.

The main program file (main module) is the input file that contains the instructions that are normally incorporated in the main routine of all programs. The actual processing to be performed by the linked program depends on what is coded in the main module. For detailed information on specifying input files, refer to Section 4.3.

2.3 SUBROUTINE LIBRARIES

The Linker supports two types of subroutine libraries: static subroutine libraries, and shared subroutine libraries. Both types are described in this section.

2.3.1 Static Subroutine Libraries

A static subroutine is one that is statically linked into the program object file at linktime. Its code and data sections are included in the output file produced by the Linker. External references to the subroutine are resolved at linktime to point to the subroutine at the address where it will be in memory when the program is loaded and executed. These addresses are fixed in the program object file and do not change when the program is loaded for execution. Hence, a link at linktime is called a static link.

Typically, static subroutines are collected into files called static subroutine libraries. The process for creating a static subroutine library using the Linker is described in Section 3.1.3. A static subroutine library file contains an entry point reference table which the Linker searches for names that match unresolved external references. Only the subroutines that satisfy external references are linked into the output file.

A static subroutine library file can also be specified as a regular input file on the INPUT screen. When this is done, the Linker links the entire contents of the file into the output file unless you explicitly exclude sections manually or by procedure specifications.

The Linker also supports the earlier type of static subroutine library, in which subroutines exist independently as separate files. When this type of library is specified, the Linker searches the library for file names that match undefined symbols. A file is linked if the file name matches an external reference.

2.3.2 Shared Subroutine Libraries

A shared subroutine is one that is dynamically linked to a program at runtime, and is shared by other programs running concurrently. A shared subroutine is not linked into a program's object file as static subroutines are. This helps reduce disk storage requirements for programs that use the same subroutines.

Shared subroutines are collected into files called shared subroutine libraries. The Linker can be used directly to create a shared subroutine library, as described in Section 3.1.3, but usually the SSL utility program is used for this purpose. For information about how to use the SSL utility, refer to the *VS System Administrator's Reference*.

A shared subroutine library file is identical in format to a static subroutine library file. They both contain an entry point reference table which the Linker searches to match external references to entry point names. The only difference between the two types of libraries is the way in which the subroutines are linked to a program.

A shared subroutine library is given a name (SSL alias) by the system administrator when it is installed on a system. This name is cataloged in a system file that describes where each shared subroutine library is located. SSL aliases are used in the Linker to specify what libraries are to be accessed by the Loader for shared subroutines called in the program.

When a shared subroutine library is specified for linking using its SSL alias, the Linker searches the entry point reference table for names that match undefined symbols. When a match is found, the Linker associates the SSL alias with the undefined symbol and places the symbol-alias pair in a special table included in the program output file. For each SSL symbol, the Linker also includes a list of all the locations in the program object file where references to the symbol are made. When the program is loaded for execution, the Loader uses this information to access the appropriate shared subroutines and to resolve all references to them in the program.

The use of shared subroutines is not limited to system subroutines; any collection of user subroutines can be organized as a shared subroutine library.

2.3.3 Programming Language Requirements for Subroutine Libraries

When you are writing subroutines in Assembly language that are intended to be used in a shared subroutine library, you must be aware of the following condition:

- Address constants that refer to shared subroutine library entry points *must always be put in static sections*, never code sections.

References to shared subroutine library entry points that are made in code sections cannot be resolved by the Linker or the Loader; they will cause address exceptions or other errors when programs make references through them.

Subroutines in a shared subroutine library are dynamically linked by the Loader when a program that refers to them is being loaded for execution. Since the subroutines can reside anywhere in memory, the Linker cannot resolve address constants that refer to shared subroutine library entry points; the Loader performs that task. However, the Loader cannot resolve address constants in code sections because the code is not modifiable at load time.

The compiled languages such as PL/I and FORTRAN adhere to the rule to always put address constants that refer to shared subroutine library entry point names in static sections. However, BASIC does not adhere to this rule for such references. Therefore, BASIC programs currently cannot reference subroutines in a shared subroutine library, nor can subroutines written in BASIC be placed in a shared subroutine library.

If you are writing subroutines to be placed in a *static subroutine library*, you can put address constants in both static sections and code sections. This is possible since the Linker includes static subroutines in the object file for the program being linked, and can resolve the address constants based on the program's base-address. However, it is recommended that you place address constants in static sections for static subroutines also. In this way, you can subsequently include the subroutines in a shared subroutine library, if desired.

2.4 THE LINK MAP

The link map contains a detailed description of the Linker run. The Linker builds each portion of the link map separately, as you provide information on the various screens that appear as a result of the options you chose. You can instruct the Linker to display a selected portion of the link map by pressing PF14 from any screen that provides that function.

If you specify YES for the MAP option at the start of the link, the Linker will print the map at the end of the link. Note, however, that the Linker builds the link map when you press PF14 regardless of your choice of the link map option. This allows you to examine the progress of the Linker run without producing a printout at the end of the link. Each time you press PF14, the Linker regenerates the portion of the link map you choose to view.

If a link map is to be printed, the Linker displays a screen that lists each portion of the link map for selection. You simply enter YES or NO for each portion, and the link map is printed accordingly. For detailed information on specifying the link map print options, refer to Section 4.4.2.

2.4.1 Link Map Topics

The link map contains the following information in the same order as it appears in this list:

1. Input log
2. Input files
3. Linked code sections
4. Linked static sections
5. Duplicate sections
6. A cross-reference list arranged in memory location order
7. A cross-reference list arranged in symbol order
8. Base-dependent code sections
9. Shared subroutine library symbols and aliases
10. Subroutine library entry point names and information
11. Undefined symbols
12. Output object file summary
13. Warning messages

Each of these topics is described in this section. Appendix A shows sample link maps.

Input Log

This section lists the linker options, input files, subroutine libraries, and output specifications. This information is printed at the beginning of the link map. It is also printed at the beginning of warning messages if a link map is not specified.

Input Files

This list is divided into two sublists: one contains the individual files input by the user, and the other contains the files retrieved from the user-specified subroutine libraries.

Linked Code Sections

The code sections are listed in the order in which they are linked. For each section, the following information is provided:

- The name of the section
- The base address where the section was relocated (its origin)
- The length of the section
- A list of entry point names including the offsets with respect to the section
- The translator name that generated the section
- The version number of the translator
- The date and time the section was translated
- The input file where the section was contained
- A flag to indicate whether the section contains symbolic data
- A flag to indicate whether there are duplicate section names

Linked Static Sections

The static sections are listed in the order in which they are linked. The information provided for each static section is the same as that described for the linked code sections, except that there is no symbolic flag.

Duplicate Sections

This list contains information about sections that have the same name occurring in more than one input file. For each duplicate section name, all the input files containing the section name are listed. The file containing the section that was selected for linking is indicated. The Linker provides the following information for each section:

- The input file name
- The section type (code or static)
- The length of the section
- The translator name that created the section
- The version number of the translator
- The date the section was created
- The time the section was created
- A flag to indicate whether the section contains symbolic data

Cross-Reference List in Location Order

This list contains symbolic references in the linked output that are arranged in memory location order. The following information is included for each memory location:

- The location address
- The section name that contains the location address constant
- The section type (code or static)
- A flag to indicate whether the address constant is R-Type (RCON) or A-Type (ACON) (Assembly language address constants)
- A flag to indicate whether the address is defined in a relocation record in the static block
- The length of the address constant (3 bytes or 4 bytes)
- The direction of relocation (positive or negative)
- The symbols referred to by the address constant. The following information is provided for each symbol:
  - The address of the symbol
  - The section where the symbol is defined
  - The section type (code or static)

Cross-Reference List in Symbol Order

This list contains symbolic references in the linked output that are arranged alphabetically in symbol order. The following information is included for each symbol:

- The name of the symbol
- The address of the symbol
- The name of the containing section and its type (code or static)
- The locations of the references. In addition, the following information is provided for each reference:
  - The address of the reference
  - The section name that contains the address constant
  - The section type (code or static)
  - A flag to indicate whether the address is an RCON or an ACON
  - A flag to indicate whether the address is defined in a relocation record in the static block
  - The length of the address constant (3 bytes or 4 bytes)
  - The direction of relocation (positive or negative)

Base-Dependent Code Sections

This list identifies all code sections in the output file that are base-dependent. Base-dependent code is code that must be executed at a certain location in the user's address space in order to run correctly.

An output file that contains base-dependent code cannot be used as a shared subroutine library (SSL). For more information, refer to Appendix C.

Shared Subroutine Library Symbols and Aliases

This list contains all shared subroutine library symbols. For each symbol, the list provides the associated alias name and a count of the distinct alias names associated with the symbol.

Subroutine Library Entry Point Names and Information

This list contains all subroutine library entry point names for either a static or a shared subroutine library.

Undefined Symbols

This list contains all the undefined symbols that exist in the output file.

Output Object Format Statistics

This list contains the following statistics that pertain to the output object format:

- Total length of the linked code sections
- Total length of the linked static sections
- The program base-address
- The program entry-point name, address, and offset
- The program name from the Prolog block
- The program release date
- The version number assigned to the program
- The output file name and location
- The number of output records
- The object code format
- A statement to indicate whether the linked code sections are base-dependent when loaded at runtime

2.5 WARNING MESSAGES

Warning messages that pertain to the output file are printed at the end of the link map, if you specify MAP = YES on the OPTIONS screen. If you specify MAP = NO, the warning messages are placed in a print file in your spool library with a system-generated file name, unless you are running the Linker by a procedure. In this case, you can specify the name of the print file with an ENTER PRINT clause.

If the link map was not printed, the warning messages are preceded by an Input Log to identify the Linker run. Note that the WARNINGS option on the MOREOPT screen enables you to specify that no warning message print file be created. Refer to Section 4.2.2 for more information on the WARNINGS option.

2.6 THE OUTPUT FILE

When you have finished specifying all input according to the Linker options that you chose, you must name and describe the output file. The output file contains all specified components of the link. If Version 1 object code is selected as the output format, it contains the following sequence of blocks of information:

- The Prolog block
- The Code block containing the linked code sections
- The Lengths block
- The Static block containing the linked static sections
- The Module block (if created) containing information required for dynamic linking and for static subroutine file entry points
- The Symbolic block (if created) containing symbolic information for all sections
- The Linkage block (if created) containing linkage information for all sections

The output file provides all relevant linking information that collectively makes up the assembled program file, static subroutine library, or shared subroutine library. In order to name and describe the output file, the Linker displays a screen that lists a number of output file attributes. For detailed information on naming and describing the output file, refer to Section 4.4.

CHAPTER 3
LINKER FUNCTIONS AND RUNNING MODES

3.1 LINKER FUNCTIONS

The Linker provides the following basic functions:

- Links program modules
- Replaces program modules
- Creates and maintains static and shared subroutine libraries

You can run the Linker in one of the following modes:

- Interactive mode
- By procedure in the foreground
- By procedure in the background

This chapter provides conceptual information on the basic Linker functions. In addition, it describes the Linker running modes and how to initiate each mode.

3.1.1 Linking Program Modules

In modular programming, a large program is divided into a number of small, self-contained modules (program files). Each module is designed to perform a single, well-defined task. The modules are written as small subprograms or subroutines and are compiled independently of the main program, which contains the main routine.

The Linker is the vehicle for linking specified program modules into a single, assembled output file. Typically, you specify a main program file and a number of program modules as input for the link. Depending on the options specified, the Linker then creates one of the following entities:

- An executable output program file
- A static subroutine library to be used for subsequent links
- A shared subroutine library to be used for subsequent links

3.1.2 Replacing Program Modules

The Linker can be used to selectively replace one or more object modules in a program file without having to relink the whole program. This can save time for large programs where, for instance, only one module has been recompiled. Another use for selective linking is to replace an object module that does not have symbolic information, with another object module that does. This is useful for debugging purposes.

You can use a procedure to selectively replace program modules by specifying the object file(s) that contain the new section as the first input file, followed by the main program file. Note that the name of the entry point into the main program file must also be specified. Figure 3-1 illustrates how you can use a procedure to replace a program module.

```
        procedure replacep
link:   run linker
        enter options map=no, resolve=yes
        enter input file=OBJECT, library=vselobj, volume=system
        enter input file=MAIN, library=vselrun, volume=system
        enter input
        enter library
        enter output file=MAIN, library=vselrun, volume=pacman,
                     replace=yes, entry=ENTMAIN
        return code=link
```

Figure 3-1. Using a Procedure to Replace a Program Module

In addition to using a procedure to replace a program module, you can also selectively replace one or more object modules in a program file, by specifying YES for the DUPSECT option on the MOREOPT screen (Figure 4-3). This enables you to resolve duplicate section names interactively. Then, you specify the main program file as the first input file, followed by one or more new object files.

For each duplicate section encountered, the Linker will display a screen from which you can select the appropriate object file for the section to be included in the link. By choosing the new object file in each case, you will replace the old sections in the program file with the new sections that have the same names.

3.1.3 Creating and Maintaining Subroutine Libraries

The Linker creates a subroutine library file instead of a program file if you specified YES for the STATICSL option or the SHAREDSL option on the OPTIONS screen (Figure 4-2). A subroutine library (static or shared) is used in subsequent links to supply subroutines referred to by a program. The subroutines are either statically linked or dynamically linked in the program, depending on the type of subroutine library.

The Linker is also used to replace, add, or delete subroutines in an existing subroutine library. This is done by specifying the subroutine library file as an input file along with any new subroutine object files, and managing the makeup of the new subroutine library through the various Linker options.

There are two ways in which subroutines can be replaced in a subroutine library: automatic replacement or selective (manual) replacement. To replace subroutines by automatic replacement, you specify the new object files first, in order of input, followed by the subroutine library file last. The new object modules will replace the old ones in a subroutine library according to the Linker default selection rules for duplicate sections. For a detailed description of the default selection rules, refer to Appendix B.

To replace subroutines by selective replacement, you must specify YES for the DUPSECT option on the MOREOPT screen to override the default selection rules. This enables you to choose which of the duplicate sections are to be included in the link.

You delete subroutines from a subroutine library by specifying YES for the EXCLUDE option on the INPUT screen (Figure 4-4). You can add or delete entry point names by specifying YES for the ENTNAMES option on the OPTIONS screen (Figure 4-2).

After all the desired updates are completed, you replace the previous subroutine library file with the new file by giving the output file the same subroutine library name, and specifying YES for the REPLACE option on the OUTPUT screen (Figure 4-13).

3.2 RUNNING MODES

This section describes the Linker running modes and provides conceptual information on using VS procedures.

3.2.1 Running the Linker Interactively

In an interactive link, the Linker displays the screens according to the Linker options that you choose and the information that the Linker requires to perform the link. You interact with the linking process by selecting the options and entering required information on a screen-by-screen basis. The link is complete when you have selected all the options necessary for the link and provided any additional information that the Linker requires, and the Linker has built the output file.

To initiate an interactive link, follow the instructions described in Section 1.7.1.

3.2.2 Using VS Procedures to Run the Linker

In many cases, you will find it convenient to use a VS procedure to run the Linker. Procedures are special routines that invoke system functions and supply required parameter information with little or no user interaction. The VS Procedure language provides the capability to create procedures.

You can write procedures to perform a partial link or the entire link. If a procedure performs a partial link, the remainder of the link is done interactively by the user, and it must run in the foreground. If a procedure performs the entire link, it can run in the background from the Command Processor. These two methods are described in this section.

Some procedures may require user interaction at certain stages. For example, if you specify in a procedure that you want to reorder sections of the output file, the procedure pauses at the appropriate point in the link and enables you to reorder the sections. When you indicate that you have reordered the sections, the procedure resumes control.

Chapter 5 describes the Procedure language syntax for each Linker screen. For detailed information on the VS Procedure language, refer to the *VS Procedure Language Reference*.

Running the Linker by Procedure in the Foreground

To run a procedure in the foreground is to actively run the procedure as a workstation task rather than submitting it to a procedure queue to be run in the background.

To run the Linker by procedure in the foreground, press PF1 from the Command Processor menu. The system displays the Run screen, which enables you to identify the procedure.

Enter the file, library, and volume names of the procedure on the Run screen and press ENTER. The system then transfers control to the procedure, which performs the tasks that it was coded to do.

If the Linker detects an error while being run by a procedure in the foreground, it displays an error screen. The error screen states the problem and enables you to enter the correct information, and to continue the link. Since error screens have no additional capabilities other than obtaining corrected information, they are not shown in this manual.

Running the Linker by Procedure in the Background

To run a procedure in the background, run the procedure entirely without user interaction. The procedure is submitted as a background task. The system enters it on the procedure queue, and runs it when its turn comes up.

All parameters required by the Linker in a background run must be supplied by the controlling procedure, since background jobs are prohibited from any interaction with the workstation. This information includes all instructions that would normally require user interaction.

You supply parameters to a procedure by using multiple ENTER statements or default values (such as with a SET statement). You must use a SET statement within the background procedure to specify any defaults used within the background job; for example, to specify print mode defaults.

To run the Linker by procedure in the background, press PF12 from the Command Processor menu. The system displays the Submit Procedure screen, which enables you to identify the procedure, and enter scheduling and execution options. For detailed descriptions of those options, refer to the *VS System User's Introduction*.

Enter the file, library, and volume names of the procedure on the Submit Procedure screen and press ENTER. The system then enters the procedure in the next available location on the procedure queue. When its turn comes, the procedure is run in the background, transparent to the user.

If the Linker detects an error, or requires any user interaction while running in the background, the Linker prints an error message and terminates execution of the background job.

CHAPTER 4
THE LINKING PROCESS

4.1 INTRODUCTION

This chapter describes the linking process in detail and documents all Linker screens used for interactive linking. The following list summarizes the linking process.

1. Initiate the Linker (refer to Section 1.7.1)

2. Specify the Linker options to be used in the link

3. Specify the input to be used in the link

   a. Specify individual input files or choose files from a library
   b. Resolve duplicate section names (if present)
   c. Optionally exclude sections from the link
   d. Manage unopened files (if present)
   e. Manage invalid files (if present)
   f. Resolve undefined symbols (if present)

4. Name and describe the output file

   a. Specify the output file parameters
   b. Optionally assign aliases to resolve SSL symbols
   c. Optionally specify link map print options
   d. Optionally reorder program sections
   e. Optionally select subroutine library entry point names

5. Build the output file

Figure 4-1 shows the Linker screen flow.

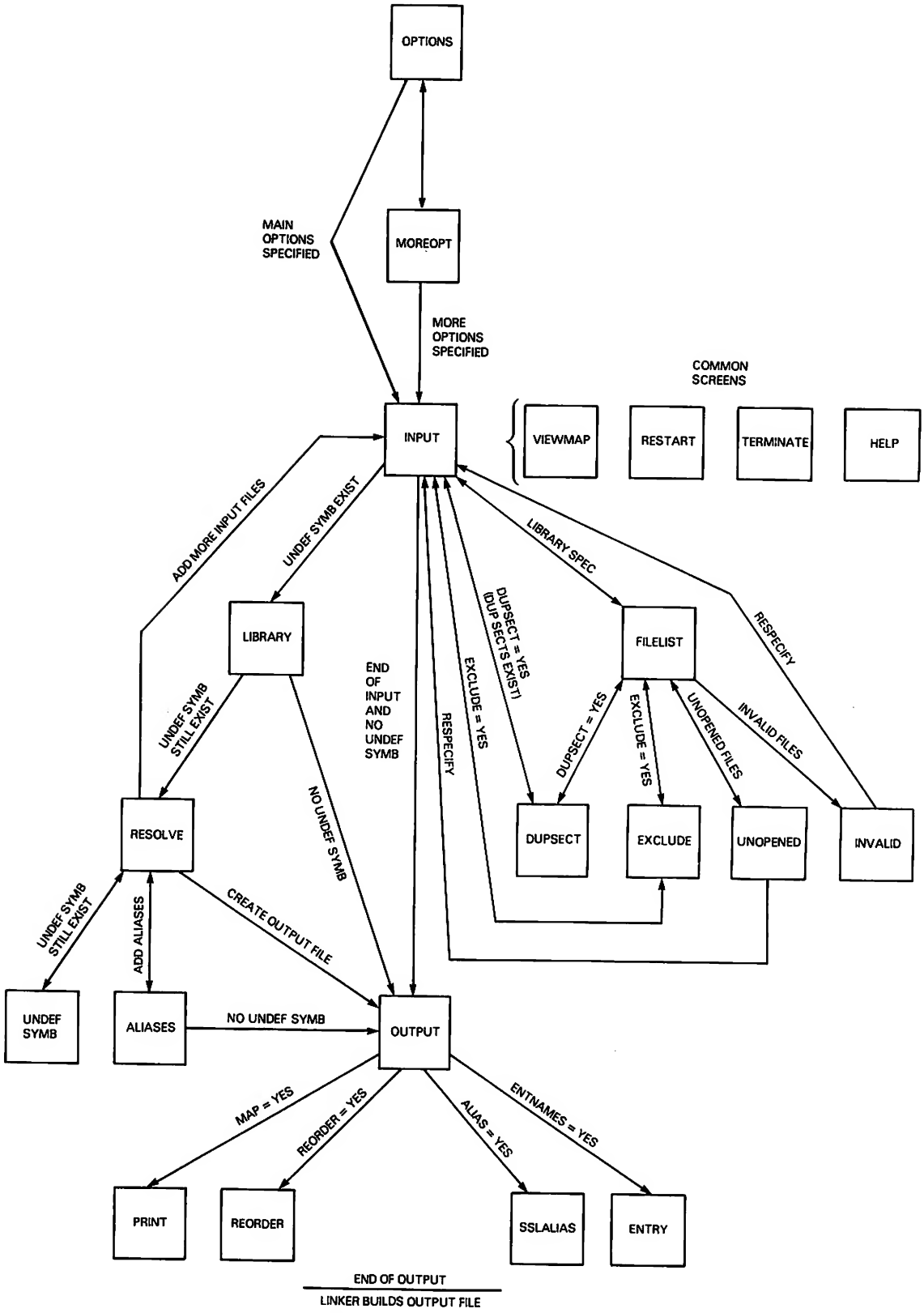

Figure 4-1. Linker Screen Flow

4.2 SPECIFYING THE LINKER OPTIONS

The first screen that the Linker displays when you run the Linker interactively is the OPTIONS screen (Figure 4-2). The OPTIONS screen lists the main Linker options (the most commonly used options), and enables you to specify those that you want to include in the current link.

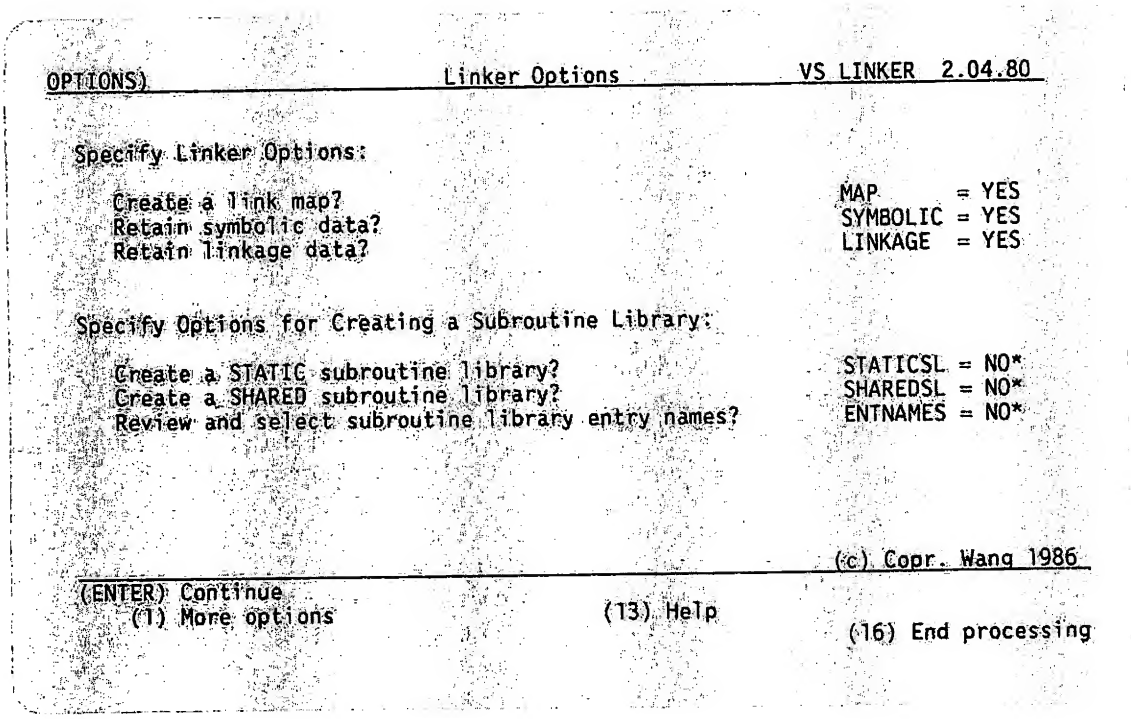

Figure 4-2. The OPTIONS Screen

The Linker options that appear on the OPTIONS screen are summarized as follows:

MAP = -- Enter YES if you want to create a link map file for the program; otherwise, enter NO. The default is YES. The contents of a link map are described in Section 2.4.

SYMBOLIC = -- Enter YES if you want to retain all symbolic data in the output file for subsequent program debugging; otherwise, enter NO. The default is YES. IF you enter NO, all symbolic data is excluded from the output file, thereby disabling symbolic debugging for the program. This option is described in Section 4.2.1.

The SYMBOLIC option can be overridden on a file-by-file basis from the INPUT screen (Figure 4-4) and the FILELIST screen (Figure 4-5).

LINKAGE = -- Enter YES if you want to retain all linkage data throughout the program for subsequent program linking and debugging. Enter NO if you want to exclude the linkage data. The default is YES. Note that programs without linkage data cannot be relinked or used as input to another link.

The last three options pertain only to creating a subroutine library:

STATICSL = -- Enter YES if you want to make the output file a static subroutine library to be used in subsequent program links; otherwise, enter NO. The default is NO. This function is described in Section 3.1.3.

SHAREDSL = -- Enter YES if you want to make the output file a shared subroutine library to be used in subsequent program links; otherwise, enter NO. The default is NO. This function is described in Section 3.1.3. Whenever you create a shared subroutine library, always check the link map for base-dependent code sections. Base-dependent code sections cannot be used in a shared subroutine library. For more information, refer to Appendix C.

ENTNAMES = -- If you specified YES for either the STATICSL or SHAREDSL options, enter YES to display a list of all subroutine library entry point names; otherwise, enter NO. The default is NO. From that list, you can select the entry point names to be included in the entry point reference table to resolve external references. This function is described in Section 4.4.5.

The functions available from the OPTIONS screen are summarized as follows:

PF Key	Description
ENTER	Continue -- When you have specified the Linker options, press ENTER to continue Linker processing.
1	More options -- Accesses the MOREOPT screen (Figure 4-3), which lists additional Linker options for selection.
13	Help -- Displays Help text for this screen. This function is described in Section 1.5.1.
16	End processing -- Enables you to terminate processing and exit the Linker. This function is described in Section 4.5.3.

To specify the options that you want to include in the link, enter YES or NO in the field beside each Linker option. To manage an additional set of less commonly used options, press PF1 to display the MOREOPT screen (Figure 4-3). The MOREOPT screen is described in Section 4.2.2. When you press ENTER from either the OPTIONS screen or the MOREOPT screen, the Linker accepts the selected options and displays the INPUT screen (Figure 4-4).

4.2.1 Managing the Symbolic Debugging Data

Symbolic debugging information describes all symbolic code in the program, and enables the Debugger to accept references to data values by symbolic name. This information is contained within the input files produced by the assembler or compiler.

The Linker provides the option to retain or exclude the symbolic debugging information on the INPUT screen (Figure 4-4) and on the FILELIST screen (Figure 4-5). If you specify YES for the SYMBOLIC option on either screen, the Linker retains the symbolic debugging data for the displayed file, thereby providing full symbolic debugging support for that file. If you specify NO for the SYMBOLIC option, the Linker removes symbolic debugging information from the displayed file, thereby disabling symbolic debugging for that file. Machine-level debugging is still available for files when NO is specified for the SYMBOLIC option.

After a program is debugged, it is advantageous to remove symbolic information to reduce disk storage requirements for the program. Removing symbolic information also provides a level of security to prevent users from tampering with the internal workings of the program.

4.2.2 Specifying Additional Linker Options

If you press PF1 from the OPTIONS screen (Figure 4-2) the Linker displays the MOREOPT screen (Figure 4-3). The MOREOPT screen lists additional, less commonly used Linker options for selection.

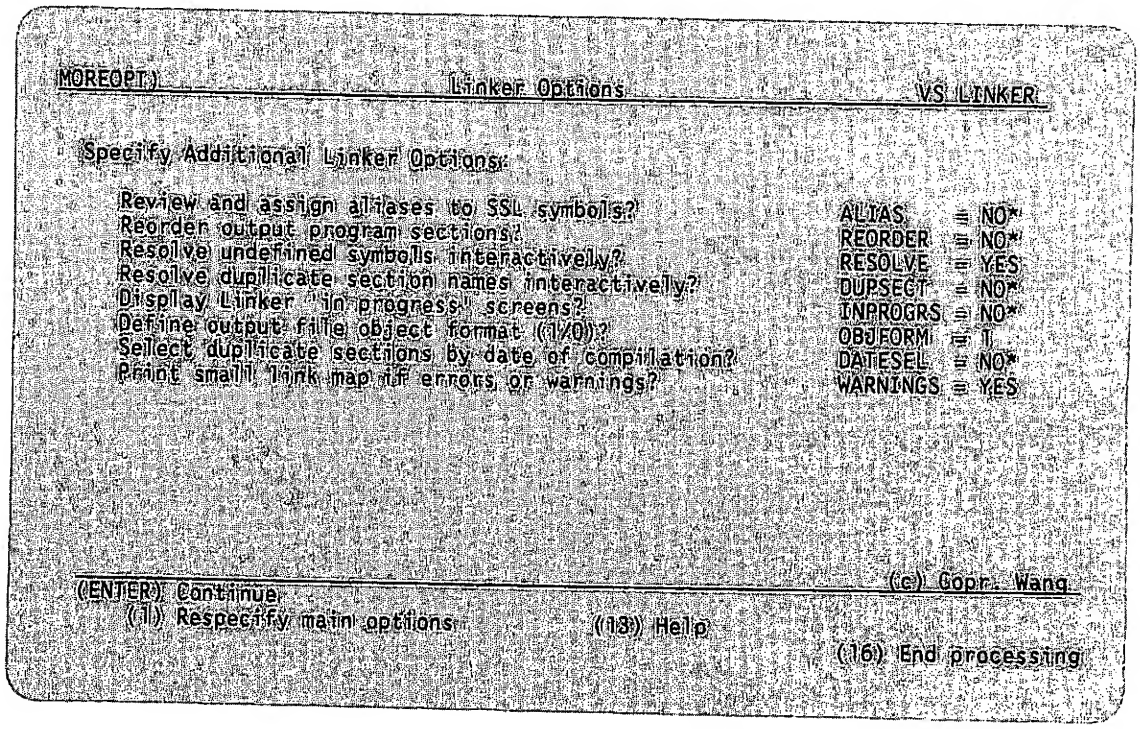

Figure 4-3. The MOREOPT Screen

The Linker options that appear on the MOREOPT screen are summarized as follows:

ALIAS = -- Enter YES to designate or reassign SSL aliases for shared subroutine library symbols in the linked output; otherwise, enter NO. The default is NO. If you enter YES, the Linker displays the SSLALIAS screen (Figure 4-18), which lists all undefined symbols with the corresponding SSL alias (if present) of each symbol. The SSLALIAS screen enables you to modify current assignments and make additional assignments. This function is also available from the OUTPUT screen (Figure 4-13), and is described in Section 4.4.4.

REORDER = -- Enter YES to change the order of code and static sections in the linked output; otherwise, enter NO. The default is NO. This function is also available from the OUTPUT screen (Figure 4-13), and is described in Section 4.4.3.

RESOLVE = -- Enter YES if you want the RESOLVE screen (Figure 4-11) to be displayed when undefined symbols exist; otherwise, enter NO. The default is YES. The RESOLVE screen, described in Section 4.3.8, provides options that enable you to manage the undefined symbols.

DUPSECT = -- Enter YES if you want to override the default selection rules and resolve duplicate section names interactively; otherwise, enter NO. The default is NO. The default selection rules apply when two or more sections from different program files have the same name. Since section names must be unique in the output file, the rules determine which section to use. The default selection rules are described in Appendix B.

If you enter YES, the Linker displays, in sets of two, any duplicate sections that occur and enables you to choose which section to include. This function is described in Section 4.3.5.

INPROGRS = -- Enter YES to display the Linker "in progress" screens; otherwise, enter NO. The default is NO. These screens keep you informed of the processing that the Linker is currently performing.

OBJFORM = -- This option specifies the object code format for the output file. The default is 1. If you specify Format 0, the output file is produced in a format that is recognized and used by Release 6 and earlier versions of the operating system. If you specify Format 1, the output file is produced in a format that was introduced with Release 7.

The Release 7 operating system executes both Format 0 and Format 1 object code. Release 6 and earlier operating systems recognize only Format 0: They cannot load and execute Format 1 object code.

The executable file produced with object format 0 has the same format as executable files created with the 6 series Linker. When object format 0 is selected, no options pertaining to SSLs or static libraries appear on the Linker screens. The Alias field on the Output screen is also disabled. The format of the link map will remain unchanged, except that the sections for SSLs are omitted. If you specify OBJFORM = 0 and SHARESL = YES, on the MOREOPT screen, the system displays the following message:

"Shared Subroutine Library can not be created in object format 0"

DATESEL = -- The default value for this field is NO. If you enter a value of YES for this field, the linker will resolve duplicate sections by date of compilation. Refer to Appendix B for detailed information about how the linker resolves duplicate sections.

WARNINGS = -- This field enables you to specify whether or not you want the system to generate a print file containing a list of warning messages. The default value for this field is YES. If you enter a value of NO for this field, the linker will not create the warning message print file.

To specify the additional options that you want to include in the link, enter YES or NO in the field beside each Linker option and press ENTER. The Linker then accepts the selected options and displays the INPUT screen (Figure 4-4).

The functions available from the MOREOPT screen are summarized as follows:

PF Key	Description
ENTER	Continue -- After you have specified the additional Linker options, press ENTER to continue Linker processing.
1	Respecify main options -- Returns to the OPTIONS screen and enables you to respecify the main Linker options.
13	Help -- Displays Help text for this screen. This function is described in Section 1.5.1.
16	End processing -- Enables you to terminate processing and exit the Linker. This function is described in Section 4.5.3.

4.3 SPECIFYING INPUT

When you press ENTER from the OPTIONS screen (Figure 4-2) or the MOREOPT screen (Figure 4-3), the Linker displays the INPUT screen (Figure 4-4). The INPUT screen enables you to specify one or more input files to link. It accepts one file or library specification at a time. If you specify a library, the FILELIST screen, which lists all the files in the library, is displayed. From this list you can select the files that you want linked. The Linker displays the INPUT screen repeatedly until you indicate that there is no more input. You do this by leaving both the Library and File fields blank and pressing ENTER.

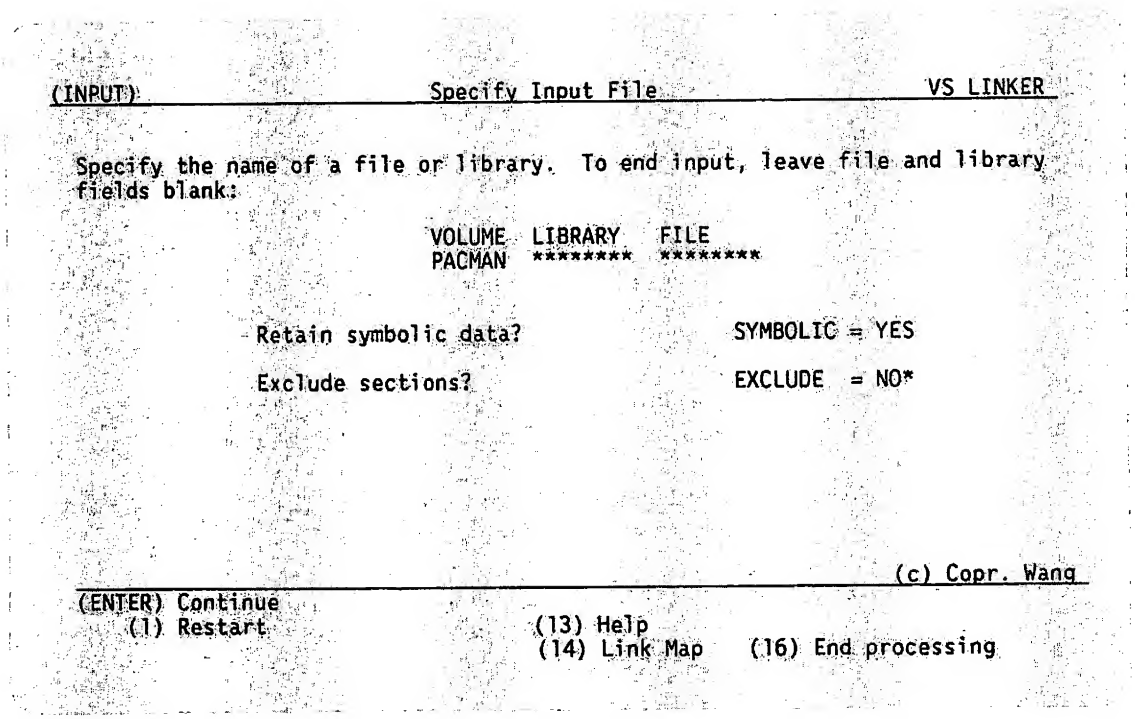

Figure 4-4. The INPUT Screen

The options that appear on the INPUT screen are summarized as follows:

Input File or Input Library = -- Enter the volume, library, and file names of the input file(s) to be linked, or specify an input library by entering only the volume and library names. If you specify a library, the Linker displays all the files in the library from which you can select the ones to be linked.

SYMBOLIC = -- Enter YES to retain the symbolic data for the displayed file in the output file; otherwise, enter NO. The value chosen for the SYMBOLIC option on the OPTIONS screen appears as the default.

EXCLUDE = -- Enter YES to interactively exclude specified sections of the input file from the linking process; otherwise, enter NO. This function is described in Section 4.3.4.

The functions available from the INPUT screen are summarized as follows:

PF Key	Description
ENTER	Continue -- Accepts the input options for the displayed file. When you have specified all input files, leave the Library and File fields blank and press ENTER. This signals the Linker that all input files have been specified and to continue Linker processing.
1	Restart -- Enables you to erase all input thus far and start the link again. This function is described in Section 4.5.2.
13	Help -- Displays Help text for the INPUT screen. This function is described in Section 1.5.1.
14	Link Map -- Enables you to display portions of the current developmental state of the link map. This function is described in Section 4.5.1.
16	End processing -- Enables you to terminate processing and exit the Linker. This function is described in Section 4.5.3.

When you have specified all input files, leave the Library and File fields on the INPUT screen blank and press ENTER. This signals the Linker that all input files have been specified and to continue Linker processing. The Linker proceeds to the next appropriate screen according to the chosen Linker options.

4.3.1 Specifying a Library

Whenever you specify a library of files (by entering only the volume and library names) on the INPUT screen (Figure 4-4), the Linker displays the FILELIST screen (Figure 4-5). The FILELIST screen lists all the program files in the specified library, and enables you to choose the files that you want to link. The SYMBOLIC and EXCLUDE options for files listed on the FILELIST screen contain the same values that appear for those options on the INPUT screen. You can accept or modify these values for each file you select.

The FILELIST screen is displayed in the foreground only. Note that any files in a specified library that have already been input to the Linker will not be listed.

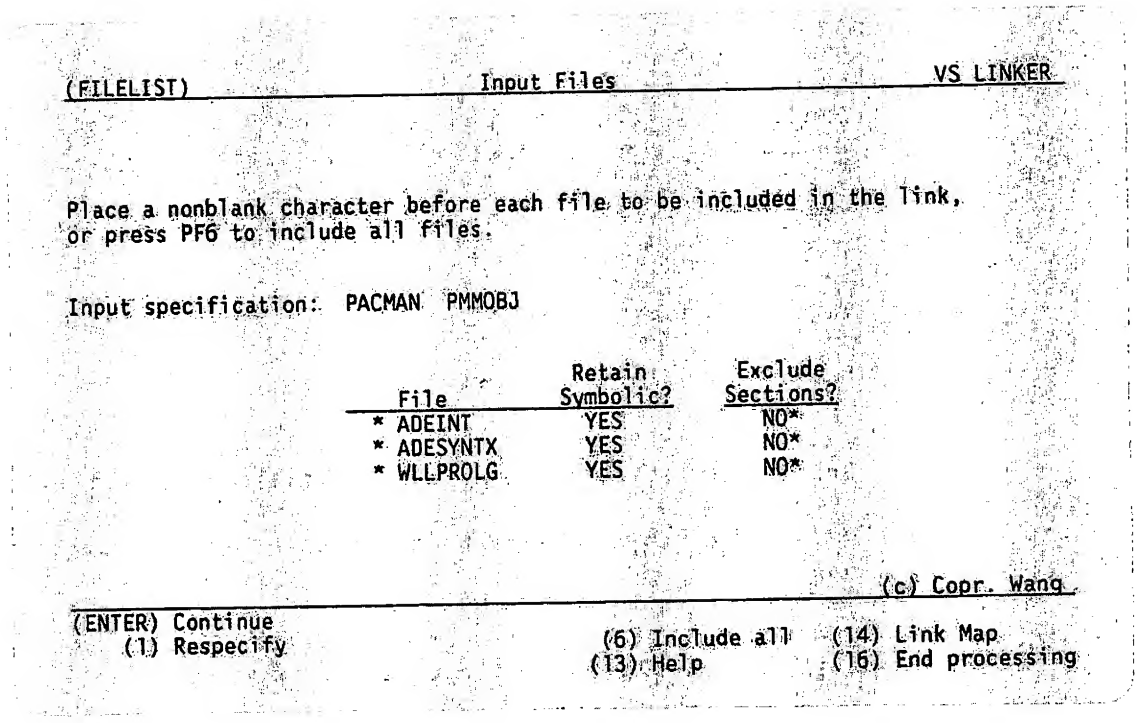

Figure 4-5. The FILELIST Screen

The options that appear on the FILELIST screen are summarized as follows:

Input Specification -- The name of the volume and library entered on the INPUT screen. If you press PF1, the Linker returns to the INPUT screen, enabling you to respecify input.

File -- The file names are listed in this section. Enter a nonblank character in the pseudoblank preceding each file that you want to link, or press PF6 to include all files.

Retain Symbolic? -- Enter YES to retain, in the output file, the symbolic data for the corresponding input file; otherwise, enter NO.

Exclude Sections? -- Enter YES to interactively exclude specified sections of the corresponding input file from the linking process; otherwise, enter NO.

The functions available from the FILELIST screen are summarized in the following list. Note that First, Last, Prev, and Next appear on the FILELIST screen only when there are more than ten files in the specified library.

PF Key	Description
ENTER	Continue -- Accepts the files and options chosen from the FILELIST screen and continues Linker processing.
1	Respecify -- Returns to the INPUT screen, enabling you to respecify input.
2	First -- Displays the first ten file names and options.
3	Last -- Displays the last ten or fewer file names and options.
4	Prev -- Displays the previous ten file names and options.
5	Next -- Displays the next ten or fewer file names and options.
6	Include all -- Marks all program files in the library with an X and displays the first ten or fewer files.
13	Help -- Displays Help text for the FILELIST screen. This function is described in Section 1.5.1.
14	Link Map -- Enables you to display portions of the current developmental state of the link map. This function is described in Section 4.5.1.
16	End processing -- Enables you to terminate processing and exit the Linker. This function is described in Section 4.5.3.

To select files from the input files list, mark the pseudoblank preceding the file name with a nonblank character, or press PF6 to select all files. Then, respond with YES or NO to the Retain Symbolic and Exclude Sections options for each file selected. If you specify YES to the EXCLUDE option for a file, the Linker displays the EXCLUDE screen (Figure 4-8).

When you have specified all files and options desired, press ENTER to continue Linker processing. If you specified YES for the DUPSECT option on the MOREOPT screen and duplicate sections exist within the files selected, the Linker displays the DUPSECT screen (Figure 4-9). Otherwise, the Linker continues processing according to the chosen Linker options.

4.3.2 Managing Unopened Files

The Linker displays the UNOPENED screen (Figure 4-6) if it is unable to open any of the files specified on the FILELIST screen (Figure 4-5). The UNOPENED screen displays those files that remain unopened, and the reason why they could not be opened, and enables you to specify how to manage them. The total number of unopened files is given below the list of file names.

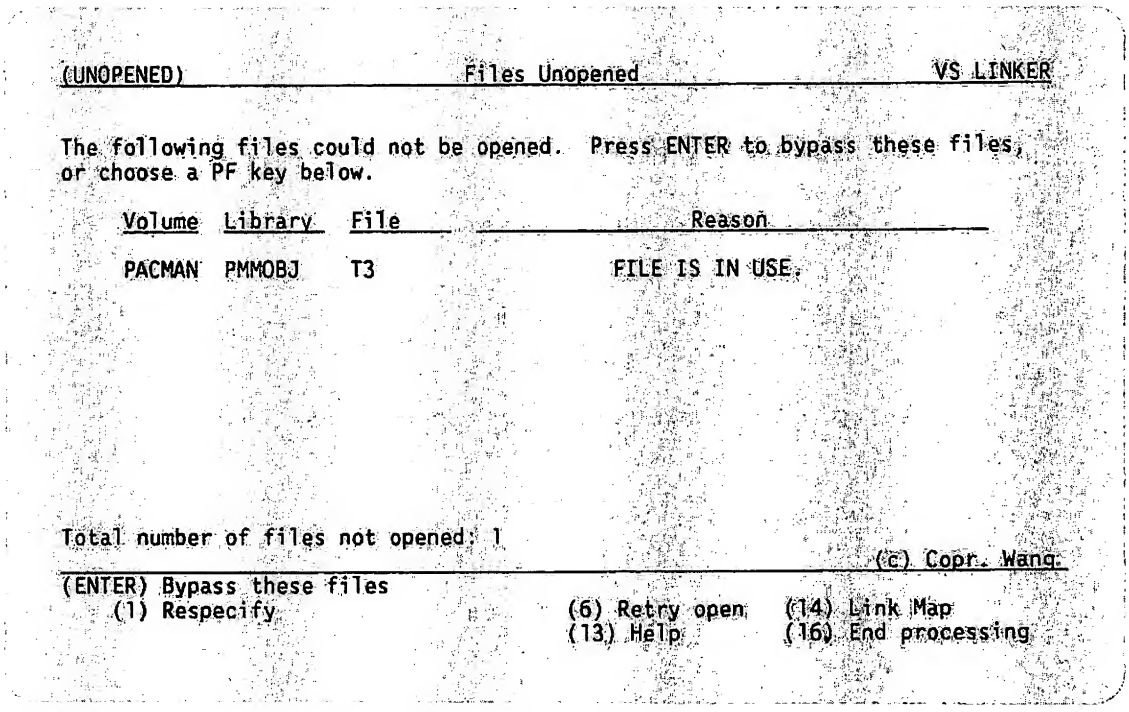

Figure 4-6. The UNOPENED Screen

The functions available from the UNOPENED screen are summarized in the following list. Note that First, Last, Prev, and Next appear on the UNOPENED screen only when there are more than ten unopened files.

PF Key	Description
ENTER	Bypass these files -- Instructs the Linker to ignore the unopened files and continue processing.
1	Respecify -- Returns to the INPUT screen, enabling you to respecify input. All files that were opened for the currently displayed input specification are closed and all associated data is erased.
2	First -- Displays the first ten unopened file names and options.

PF Key	Description
3	Last -- Displays the last ten or fewer unopened file names and options.
4	Prev -- Displays the previous ten unopened file names.
5	Next -- Displays the next ten or fewer unopened file names.
6	Retry open -- Attempts again to open all unopened files.
13	Help -- Displays Help text for the UNOPENED screen. This function is described in Section 1.5.1.
14	Link Map -- Enables you to display portions of the current developmental state of the link map. This function is described in Section 4.5.1.
16	End processing -- Enables you to terminate processing and exit the Linker. This function is described in Section 4.5.3.

The Linker provides the following alternatives for managing unopened files:

Bypass the unopened files -- If you press ENTER, the Linker ignores the unopened files and continues processing.

Respecify the input specification -- If you press PF1, the Linker returns to the INPUT screen, enabling you to respecify input. All files that were opened for the currently displayed input string are closed and all associated data is erased.

Enter the new input specification on the INPUT screen and press ENTER. The Linker continues processing as described for the INPUT screen.

Retry the open -- If you press PF6, the Linker tries again to open the unopened files.

When all program files in question have been either successfully opened or bypassed, the Linker continues processing according to the chosen Linker options.

4.3.3 Managing Invalid Files

The Linker checks each file selected from the FILELIST screen (Figure 4-5) to determine if linkage data is present. If the Linker finds a file that does not have linkage data, it displays the INVALID screen (Figure 4-7). This screen shows the volume, library, and file names of the invalid file and enables you to either bypass the file and continue processing the remaining selected files, or return to the INPUT screen to respecify the input library.

The INVALID screen is also displayed if the Linker is being run by a procedure in the foreground and an error occurs when processing an input file. An error message describing the problem appears at the top of the screen. When this happens you can either bypass the file and continue Linker processing or return to the INPUT screen to respecify the input file.

If the Linker encounters an invalid input file while being run by a procedure in the background, an error message is printed and the Linker is canceled. The INVALID screen is not displayed in this case.

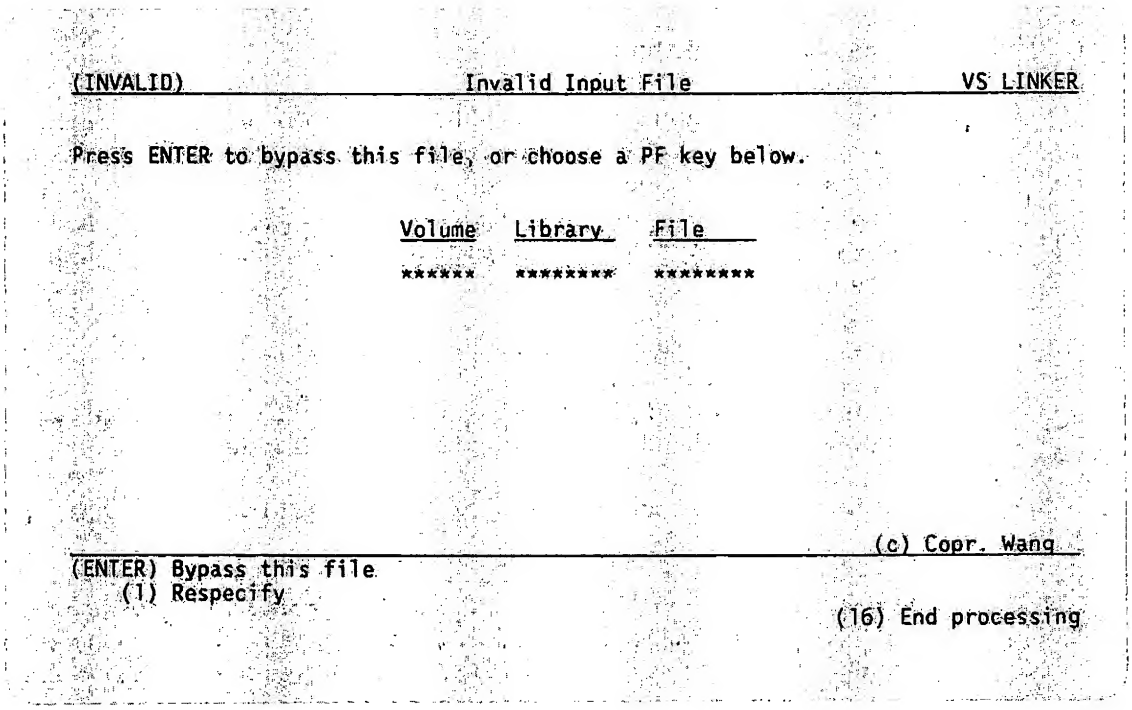

Figure 4-7. The INVALID Screen

The functions available from the INVALID screen are summarized as follows:

PF Key	Description
ENTER	Bypass this file -- Bypasses the displayed file and continues to process the remainder of the files. The Linker excludes the bypassed file when it builds the output file.
1	Respecify -- If the file was selected from the FILELIST screen, the Linker cancels all file selections on that screen and returns to the INPUT screen. All input entered prior to the canceled selections remains in effect. If running from a procedure, this function returns to the INPUT screen and allows you to change the file specification. No other files are affected.
16	End processing -- Enables you to terminate processing and exit the Linker. This function is described in Section 4.5.3.

4.3.4 Excluding Sections from the Link

For each specified input file, the Linker enables you to optionally exclude one or more sections from the linking process. Possible reasons for excluding sections are as follows:

- A number of sections in a particular input file are not needed for the link.
- Isolated cases of name conflicts may exist.
- A section is part of a shared subroutine library and will be dynamically resolved when the program is loaded.
- A section has been replaced by another section with a different name.

The Linker immediately displays the EXCLUDE screen (Figure 4-8) if you specify YES for the EXCLUDE option for a specified source file on the INPUT screen (Figure 4-4) or FILELIST screen (Figure 4-5). The EXCLUDE screen shows all the sections of the displayed file, 20 at a time, and enables you to select sections to be excluded from the output file. The number of remaining sections is displayed below the list.

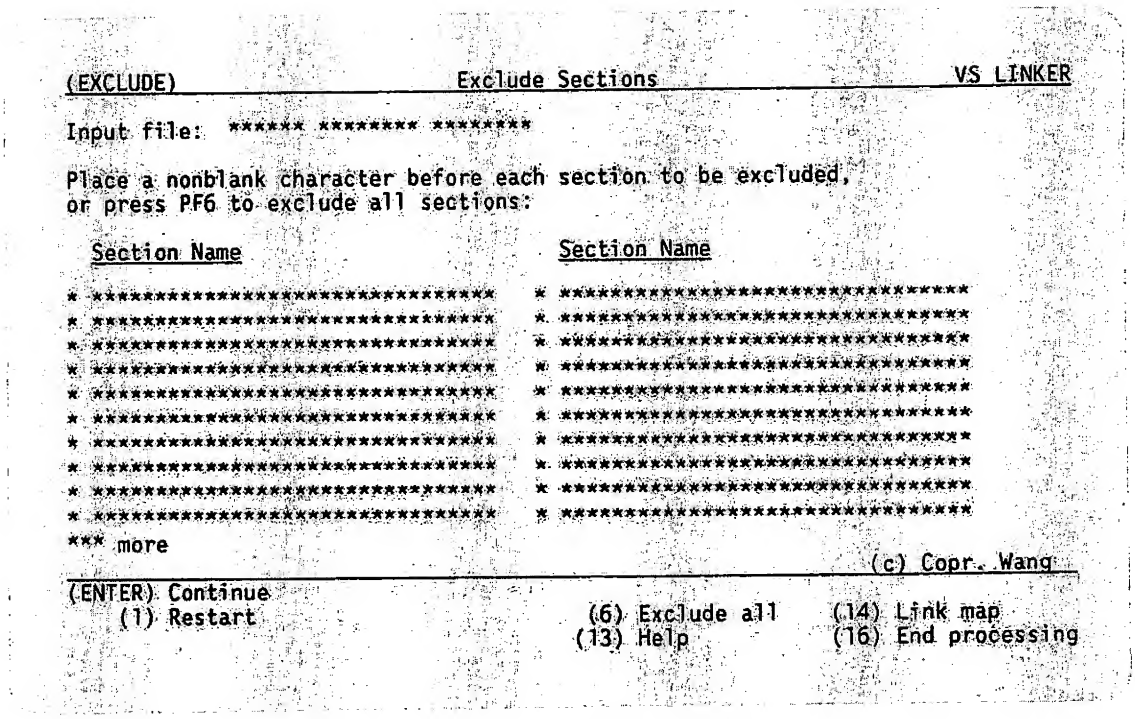

Figure 4-8. The EXCLUDE Screen

The functions available from the EXCLUDE screen are summarized in the following list. Note that First, Last, Prev, and Next appear on the EXCLUDE screen only when there are more than 20 sections in the displayed file.

PF Key	Description
ENTER	Continue -- Accepts the chosen sections to be excluded from the displayed file and continues Linker processing.
1	Restart -- Enables you to erase all input thus far and start the link again. This function is described in Section 4.5.2.
2	First -- Displays the first 20 sections.
3	Last -- Displays the last 20 or fewer sections.
4	Prev -- Displays the previous 20 sections.
5	Next -- Displays the next 20 or fewer sections.

PF Key	Description
6	Exclude all -- Automatically marks, with an X, all sections in the displayed file to be excluded from the output file; then displays the first 20 or fewer section names.
13	Help -- Displays Help text for the EXCLUDE screen. This function is described in Section 1.5.1.
14	Link Map -- Enables you to display portions of the current developmental state of the link map. This function is described in Section 4.5.1.
16	End processing -- Enables you to terminate processing and exit the Linker. This function is described in Section 4.5.3.

To exclude sections in the displayed file from the output file, mark the pseudoblanks preceding the section names with a nonblank character, or press PF6 to exclude all sections.

When you have selected all the sections to be excluded from the displayed file, press ENTER to continue Linker processing. The Linker excludes the indicated sections and returns to the INPUT screen (Figure 4-4).

4.3.5 Resolving Duplicate Section Names

Two or more sections from different input files are duplicate sections if they have the same name. Only one occurrence of a section can exist in the output file, regardless of the number of input occurrences of the section. The default selection rules (Appendix B) describe how the Linker selects a section to be included in the output file from sections with the same name.

If you specified YES to the DUPSECT option on the MOREOPT screen (Figure 4-3), you have disabled the default selection rules. If duplicate sections occur, the Linker displays the DUPSECT screen (Figure 4-9), so that you can select which of the sections is to be included in the link. If you are running the Linker by procedure, the OVERRIDE screen (refer to Section 5.10) replaces the DUPSECT screen.

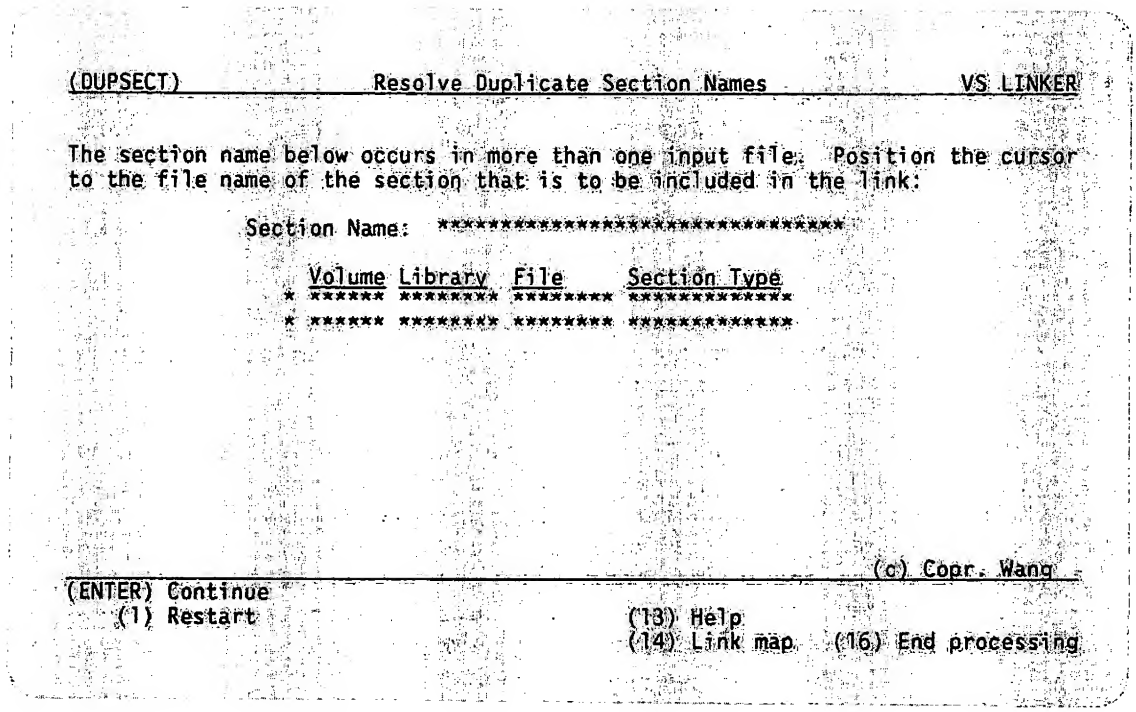

Figure 4-9. The DUPSECT Screen

The functions available from the DUPSECT screen are summarized as follows:

PF Key	Description
ENTER	Continue -- Accepts the section choice and continues Linker processing.
1	Restart -- Enables you to erase all input entered thus far and start the link again. This function is described in Section 4.5.2.
13	Help -- Displays Help text for the DUPSECT screen. This function is described in Section 1.5.1.
14	Link Map -- Enables you to display portions of the current developmental state of the link map. This function is described in Section 4.5.1.

PF Key	Description
16	End processing -- Enables you to terminate processing and exit the Linker. This function is described in Section 4.5.3.

For each duplicate section name the following information is displayed:

- The section name
- The file names and locations (library and volume) of two of the files that contain the section
- The section type (code or static) of each of the two sections

To select the section to be included in the link, position the cursor at the pseudoblank preceding the file name of the desired section and press ENTER. If there are still more sections with the same name, another file name and location and its section type is displayed along with the section that you just selected. Again, you must choose between the two. This process continues until one section has been chosen from all the duplicates. The Linker then continues processing.

4.3.6 Resolution Process for Undefined Symbols

Undefined symbols may exist after you have finished specifying input for the link. A symbol is termed undefined if it does not correspond to an included section name or an entry point name in an included section. A symbolic reference is termed unresolved if the symbol to which it refers is undefined.

An undefined symbol can be one of the following types:

- A user-specified symbol that must be resolved statically by the Linker at linktime
- A shared subroutine library symbol that must be resolved dynamically at runtime

The Linker performs the following steps to resolve undefined user-specified and shared subroutine library symbols. These steps are described in greater detail in the screen sections that follow:

1. The Linker first searches all the input files for matching entry point names and resolves the symbolic references to the corresponding entry point locations. SSL symbols with aliases assigned in a previous Linker run are treated similarly: If matching entry point names are found, the references are resolved to the entry point addresses statically, and the information concerning that dynamic (SSL) reference is discarded.

2. If undefined symbols (including shared subroutine library symbols) remain after searching all the input files, the Linker displays the LIBRARY screen (Figure 4-10). The LIBRARY screen enables you to specify static subroutine libraries for the Linker to search to resolve undefined symbols. The Linker searches the libraries in the order specified. If any matching entry point names are found, the corresponding sections are retrieved and the symbolic references are resolved to the entry point locations. Shared subroutine library symbols are resolved in the same manner; thus static subroutines can be substituted for shared subroutines.

3. If you are running the Linker by a procedure and you specified SSL aliases in an ENTER ALIASES clause, and if undefined symbols remain after Step 2, the Linker next attempts to resolve the undefined symbols by searching through the shared subroutine libraries named in the ENTER ALIASES clause. The Linker first verifies that the libraries exist on the system and then it scans the entry point reference tables, looking for names that match the undefined symbols. If any are found, the Linker assigns the corresponding SSL aliases to the symbols. The Linker then assumes that these symbols will be resolved at runtime and does not report them as undefined.

4. If the RESOLVE option is YES and there are undefined symbols remaining that do not have SSL aliases assigned, the Linker displays the RESOLVE screen (Figure 4-11). The RESOLVE screen allows you to choose one of the following options for managing the undefined symbols:

   - Create the output file with undefined symbols
   - Add more input files
   - Verify the current list of shared subroutine library aliases and optionally add more
   - Display undefined symbols

If you choose to verify SSL aliases when there are undefined symbols, the Linker displays the ALIASES screen (Figure 4-12). This screen lists all the SSL aliases that are currently assigned to SSL symbols, and also any SSL aliases that were entered by procedure specification. Up to ten aliases are displayed. You can add more SSL aliases to the list and have the Linker verify and search them for entry point names, repeating the process described in Step 3, and, in this way, resolve the remaining undefined symbols.

Another way in which you can assign SSL aliases to undefined symbols is through the SSLALIAS screen (Figure 4-18). This screen is displayed when ALIAS = YES is specified on the OUTPUT screen (Figure 4-13). The SSLALIAS screen lists all undefined symbols and the currently assigned SSL alias, if any, for each symbol. You can accept or change the current assignments, and make additional SSL alias assignments. This differs from the ALIASES screen in that you, rather than the Linker, make the assignment for each symbol.

4.3.7 Resolving Undefined Symbols by the LIBRARY Screen

The first screen that the Linker displays when undefined symbols exist is the LIBRARY screen (Figure 4-10). The LIBRARY screen enables you to specify subroutine libraries for the Linker to search to resolve undefined symbols. The primary use is to specify static subroutine libraries, but shared subroutine libraries can also be specified.

Subroutines found that match undefined symbols are statically linked into the program. You can specify an unlimited number of subroutine libraries, eight at a time.

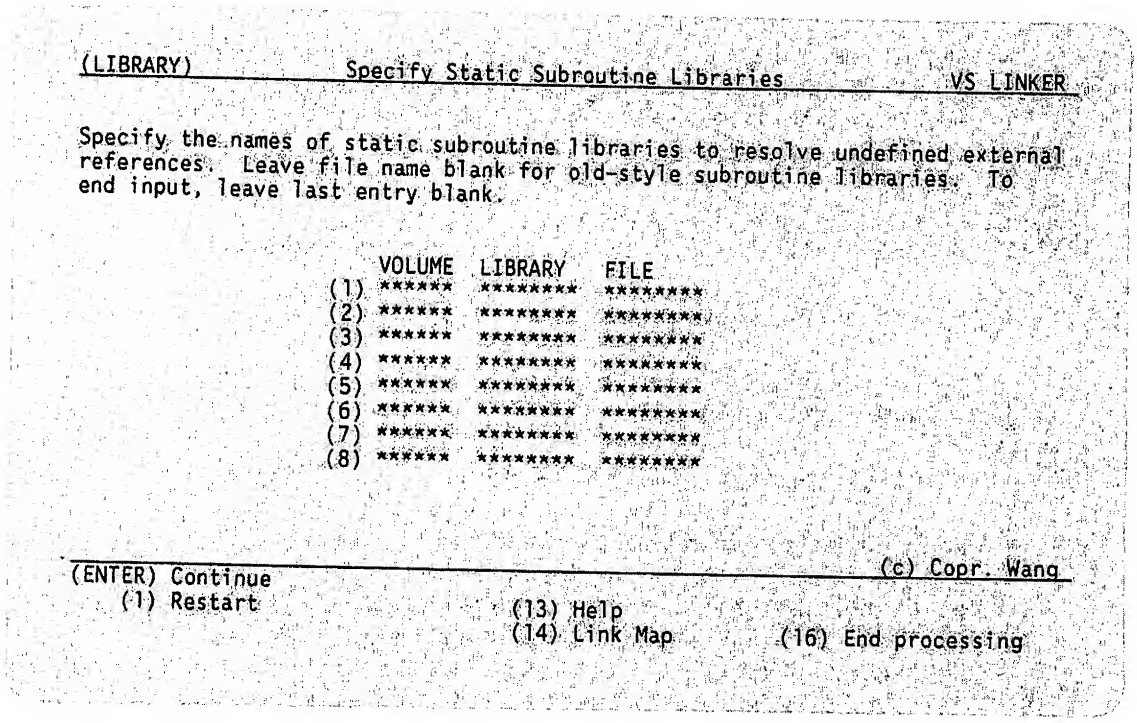

Figure 4-10. The LIBRARY Screen

The functions available from the LIBRARY screen are summarized as follows:

PF Key	Description
ENTER	Continue -- Accepts the list of subroutine libraries and continues Linker processing.
1	Restart -- Enables you to erase all input entered thus far and start the link again. This function is described in Section 4.5.2.
13	Help -- Displays Help text for the LIBRARY screen. This function is described in Section 1.5.1.
14	Link Map -- Enables you to display portions of the current developmental state of the link map. This function is described in Section 4.5.1.
16	End processing -- Enables you to terminate processing and exit the Linker. This function is described in Section 4.5.3.

There are two types of static subroutine library specifications:

1. A fully qualified name that identifies a file containing a collection of subroutines. The Linker searches the file for entry point names and links only those sections that resolve undefined symbols. (This is the new type of subroutine library organization.) To specify this type of subroutine library, enter the volume, library, and file names.

2. A partially qualified name that identifies a VS library, in which subroutines exist independently as separate files. (This type of subroutine library was supported by the previous Linker.) With this type, the Linker searches the library and links the required portions of a file if the file name matches an undefined symbol. To specify this type of subroutine library, enter only the volume and library names and leave the file name blank.

Although the LIBRARY screen is mainly used for specifying static subroutine libraries, a shared subroutine library can also be specified by entering its volume, library, and file names rather than its SSL alias. When this is done, shared subroutines that resolve undefined symbols are statically linked into the program.

Any number of static subroutine libraries can be specified, up to eight at a time. The Linker redisplays the LIBRARY screen for more specifications if the eighth entry is not blank. When you have specified all static subroutine libraries, leave the eighth entry blank and press ENTER. The Linker accepts the libraries and proceeds to the next screen according to the chosen Linker options.

Note: *The symbolic option on the OPTIONS screen is effective for all sections that are linked from static subroutine libraries.*

4.3.8 Resolving Undefined Symbols by the RESOLVE Screen

If undefined symbols remain after the Linker searches the static subroutine libraries entered on the LIBRARY screen (Figure 4-10), the Linker displays the RESOLVE screen (Figure 4-11). The RESOLVE screen provides options that enable you to manage the remaining undefined symbols.

Note: *The RESOLVE screen is controlled by the RESOLVE option on the MOREOPT screen. The default for this option is YES if you invoke the LINKER directly, and NO if you invoke the Linker through a procedure in the background. The RESOLVE screen is displayed if RESOLVE = YES and there are undefined symbols; otherwise, the screen is not displayed. The RESOLVE screen is also not displayed if the Linker is running in the background.*

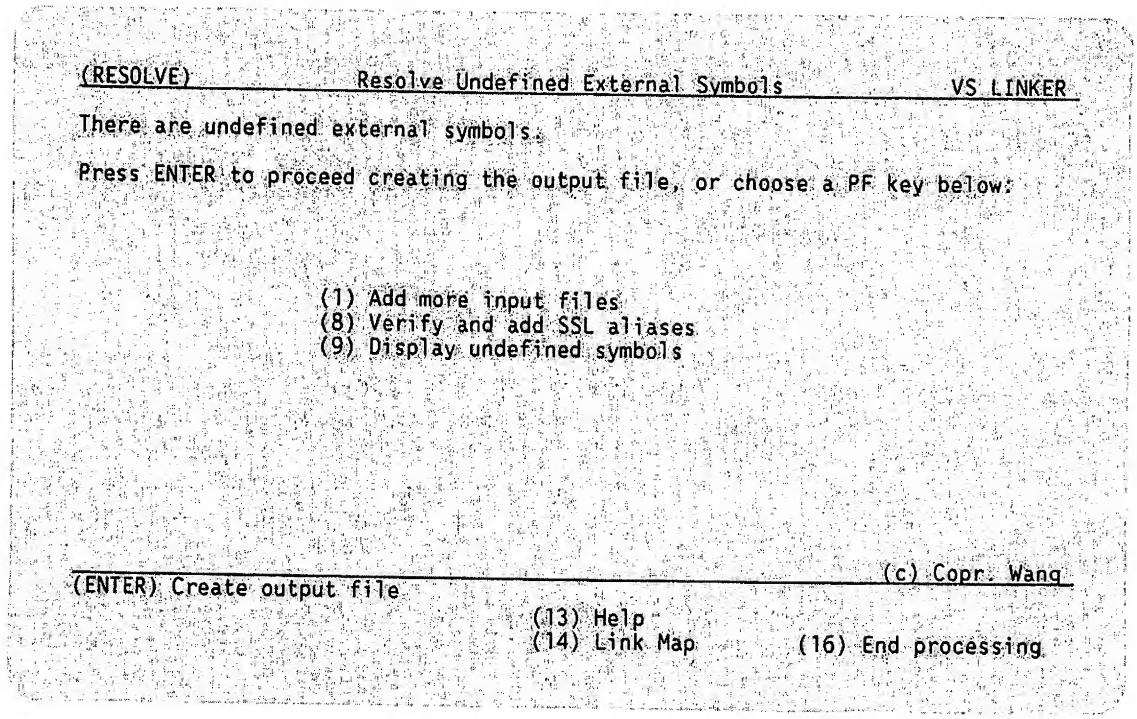

Figure 4-11. The RESOLVE Screen

The functions available from the RESOLVE screen are summarized as follows:

PF Key	Description
ENTER	Create output file -- Proceeds to the OUTPUT screen (Figure 4-13) and creates the output file including the remaining undefined symbols.
1	Add more input files -- Returns to the INPUT screen, enabling you to add more input files to resolve the remaining undefined symbols. This function is described in this section.
8	Verify and add SSL Aliases -- Displays a list of aliases currently specified for shared subroutine libraries. This function is described in this section.
9	Display undefined symbols -- Displays the undefined symbols for examination only.
13	Help -- Displays Help text for the RESOLVE screen. This function is described in Section 1.5.1.
14	Link Map -- Enables you to display portions of the current developmental state of the link map. For more information, refer to Section 4.5.1.
16	End processing -- Enables you to terminate processing and exit the Linker. This function is described in Section 4.5.3.

Adding More Input Files

To add more input files, press PF1 from the RESOLVE screen. The Linker displays the INPUT screen (Figure 4-4), and you specify the input files as described for the INPUT screen. After you have added more files, press ENTER from the INPUT screen, leaving the file and library fields blank.

If undefined symbols without SSL assignments still remain, the Linker again displays the RESOLVE screen. Otherwise, the Linker continues to process the input according to the chosen Linker options.

Verifying and Adding SSL Aliases

To verify the current list of aliases for shared subroutine libraries, and, optionally, add more aliases to resolve undefined symbols, press PF8 from the RESOLVE screen. The Linker displays the ALIASES screen (Figure 4-12), which lists all SSL aliases currently assigned to SSL symbols, and SSL aliases entered by procedure, if any.

You can add more SSL aliases to the list. Up to ten aliases can be specified (although there is no limit to the number of aliases that can actually be assigned to SSL symbols; see SSLALIAS specification, Section 4.4.4).

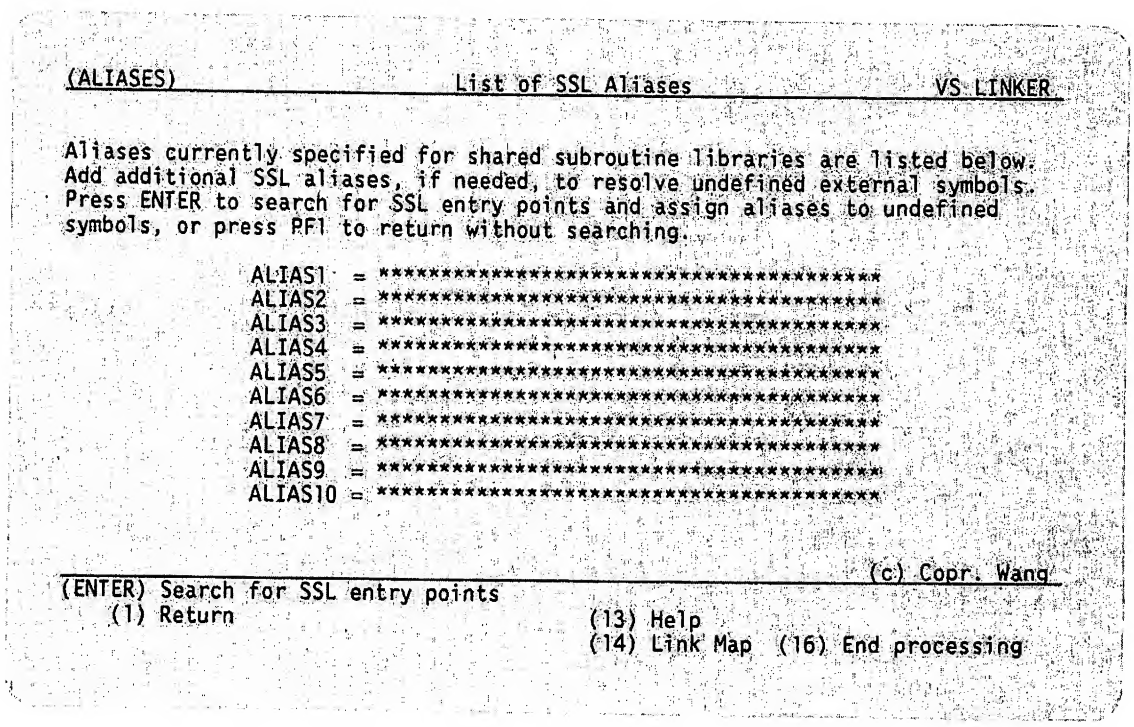

Figure 4-12. The ALIASES Screen

The functions available from the ALIASES screen are summarized as follows:

PF Key	Description
ENTER	Search for SSL entry points -- Verifies that all shared subroutine libraries exist on the system, searches each library for entry point names that match undefined symbols, makes appropriate SSL assignments to the SSL symbols, and continues Linker processing.
1	Return -- Returns to the RESOLVE screen without accepting any input. The libraries specified are not searched for entry point names.
13	Help -- Displays Help text for the ALIASES screen. This function is described in Section 1.5.1.

PF Key	Description
14	Link Map -- Enables you to display portions of the current developmental state of the link map. This function is described in Section 4.5.1.
16	End processing -- Enables you to terminate processing and exit the Linker. This function is described in Section 4.5.3.

When you press ENTER on the ALIASES screen, the Linker first verifies that the specified shared subroutine libraries exist on the system. It then searches each library for entry point names that match undefined symbols. When a match is found, the Linker assigns the corresponding SSL alias to the symbol.

If any undefined symbols without SSL assignments remain after searching the shared subroutine libraries, the Linker again displays the RESOLVE screen. If all undefined symbols have been resolved, the Linker proceeds to the next screen according to the chosen Linker options.

If the Linker informs you that an SSL alias is not installed on the system, you must either erase the entry and press ENTER again to permit the Linker to process the remaining entries, or press PF1 to return to the RESOLVE screen. If you press PF1, the Linker returns to the RESOLVE screen without processing any changes or additions to the list.

Although all the entries on the ALIASES screen can be modified, erasing or changing an existing entry does not delete or change the entry in the set of aliases maintained by the Linker. If you enter another alias for an installed SSL in place of a previous entry, the new alias is simply added to the set without replacing the original entry. Changing the list simply changes the subset of aliases that the Linker verifies the next time you press ENTER.

Aliases previously assigned to SSL symbols are not changed by the specification of libraries on this screen. In other words, a symbol that already has an alias assignment for one SSL will not be reassigned a different alias if that symbol is found in another SSL. To make changes to SSL assignments, you must reassign aliases for each symbol on the SSLALIAS screen (Figure 4-18) or by a procedure with an ENTER SSLALIAS clause (refer to Section 5.14).

4.4 NAMING AND DESCRIBING THE OUTPUT FILE

The output file contains all specified components of the link and all relevant linking information. If Version 1 object code format is selected, it contains the following sequence of blocks of information:

- The Prolog block
- The Code block containing the linked code sections
- The Lengths block
- The Static block containing the linked static sections
- The Module block (if created)
- The Symbolic block (if created)
- The Linkage block (if created)

The Linker displays the OUTPUT screen (Figure 4-13) after it has shown all of the screens that pertain to input for the chosen Linker options. The OUTPUT screen shows a list of output file attributes. From that list, you can specify the attributes of the output file.

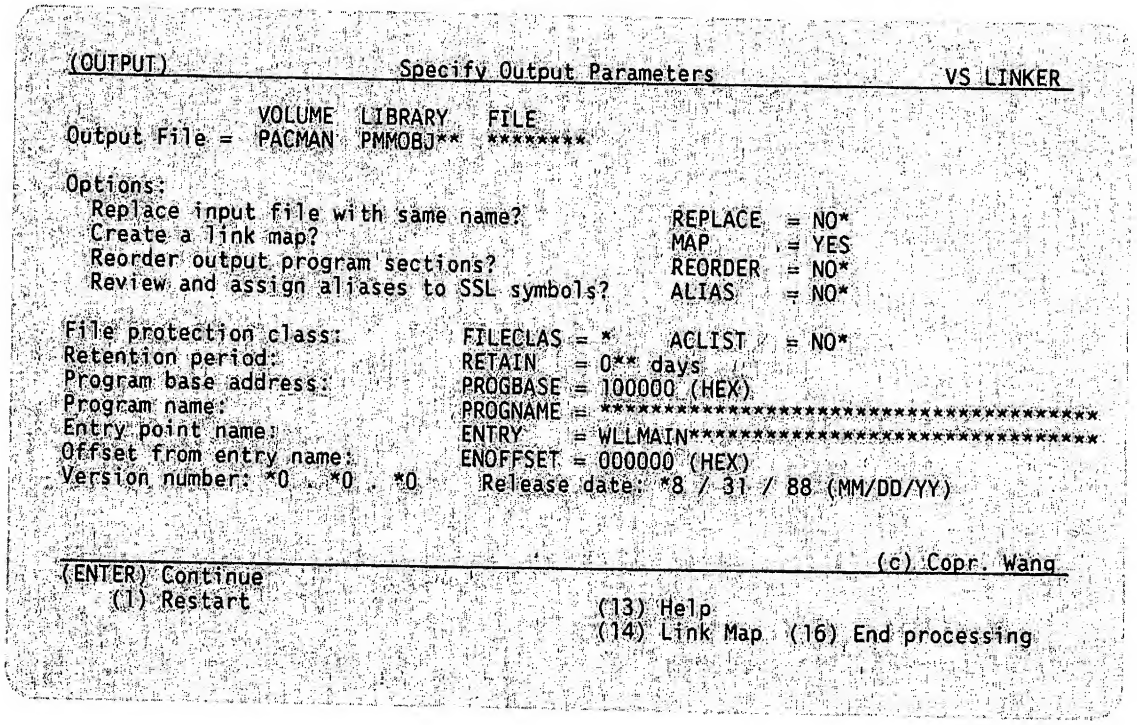

Figure 4-13. The OUTPUT Screen

The options that appear on the OUTPUT screen are summarized as follows:

Output File = -- Enter the volume, library, and file names of the output file.

REPLACE = -- Enter YES to use the input file name for the output file and delete the input file in the process; otherwise, enter NO. Program files that are currently serving as static subroutine libraries for the current Linker run are not permitted to be designated as the output file.

MAP = -- Enter YES if you want to create a link map file for the program; otherwise, enter NO. The default is YES. The contents of a link map are described in Section 2.4.

REORDER = -- Enter YES to change the order of code and static sections in the linked output; otherwise, enter NO. The default is NO. This function is described in Section 4.4.3.

ALIAS = -- Enter YES to designate or reassign SSL aliases for shared subroutine library symbols in the linked output; otherwise, enter NO. The default is NO. If you enter YES, the Linker displays the SSLALIAS screen (Figure 4-18), which lists all undefined symbols with the corresponding SSL alias (if present) of each symbol. The SSLALIAS screen enables you to modify current assignments and make additional assignments. This function is described in Section 4.4.4.

FILECLAS = -- Enter the file protection class desired. Valid entries are: blank, #, $, @, A-Z. For detailed descriptions of each file class, refer to the *VS System User's Introduction*.

ACLIST = -- Enter YES to inspect or modify your access list, or, if you do not have an access list, to create one for this file; otherwise, enter NO. The default is NO. The access list identifies the users who can access your file and the type of access granted to them. If you enter NO, your output file receives a default access list, provided you specified one previously by the Set Usage Constants option of the Command Processor menu (refer to the *VS System User's Introduction)*.

If you specified YES to the REPLACE option of this screen, your output file inherits the access list, if any, of the file it is replacing. That access list can also be modified by this option.

RETAIN = -- Enter the number of days for which you want to protect the output file from deletion. Valid entries are 0 to 999 days.

PROGBASE = -- Optionally enter a hexadecimal value for the program base address. The default is 100000. Valid entries are 000000 to FFFFFF. This is the address where the program file will be loaded at execution time.

PROGNAME = -- Optionally enter a program name of up to 40 characters for the output file. This name is in addition to the file name and location. It appears in the output file for easier identification only. It has no effect on execution.

ENTRY = -- Optionally enter the program entry point name where execution is to begin after the program is loaded. The default is the entry point name of the first input file. This field does not apply to subroutine library files.

ENOFFSET = -- Optionally enter the offset in hexadecimal from the entry point name if execution is to begin before or after the specified program entry point. The default is zero. Valid entries are 000000 to FFFFFF. This field does not apply to subroutine library files.

Version number = -- Optionally assign a version number to the program file. This number appears in the Prolog block of the output file for version identification.

Release date = -- Optionally enter the numeric values for the release date, using the first field for the month, the second field for the day, and the third field for the year. Each field should contain two digits. The date appears in the Prolog block of the output file.

The functions available from the OUTPUT screen are summarized as follows:

PF Key	Description
ENTER	Continue -- Accepts the output file options and continues Linker processing.
1	Restart -- Enables you to erase all input entered thus far and start the link again. This function is described in Section 4.5.2.
13	Help -- Displays Help text for the OUTPUT screen. This function is described in Section 1.5.1.
14	Link Map -- Enables you to display portions of the current developmental state of the link map. This function is described in Section 4.5.1.
16	End processing -- Enables you to terminate processing and exit the Linker. This function is described in Section 4.5.3.

To specify the parameters for the output file, enter the required information and your choice of options on the OUTPUT screen, and press ENTER. The Linker accepts the output parameters and proceeds to the next screen according to the chosen Linker options.

4.4.1 Inspecting, Modifying, or Creating the Access List

The Linker displays the USERS screen (Figure 4-14) followed by the GROUPS screen (Figure 4-15) if you specified YES to the ACLIST option on the OUTPUT screen. If a default access list exists, these screens contain the names and access levels of the individuals and groups who can access the file. Otherwise, they are blank.

The USERS screen specifies individual users; the GROUPS screen specifies groups of users. An access list can contain individual users, or groups, or both. You can specify a maximum of 45 individual users and 30 groups.

Group membership and group IDs are designated by the system administrator. For more information about groups, refer to the *VS System Administrator's Reference*.

If a default access list exists, you can simply inspect or modify it by changing and adding entries. If a default access list does not exist, you can create one for the output file by using these screens.

```
Wang VS GETPARM v 7                          Parameter Reference Name: USERS
                                                          Message Id: 0000
                                                           Component: GETACL
                        Information Required by PMMLOGON

*** Program File Access Control List ***

  Specify User Id(s) and Permission required to define the Access List :

  USER1  = ***   LEVEL1  = *   USER2  = ***   LEVEL2  = *
  USER3  = ***   LEVEL3  = *   USER4  = ***   LEVEL4  = *
  USER5  = ***   LEVEL5  = *   USER6  = ***   LEVEL6  = *
  USER7  = ***   LEVEL7  = *   USER8  = ***   LEVEL8  = *
  USER9  = ***   LEVEL9  = *   USER10 = ***   LEVEL10 = *
  USER11 = ***   LEVEL11 = *   USER12 = ***   LEVEL12 = *
  USER13 = ***   LEVEL13 = *   USER14 = ***   LEVEL14 = *
  USER15 = ***   LEVEL15 = *
                                               MORE   = NO*
          Press : (ENTER) Validate entries and Continue
                  (1)  to re-start data entry
                  (16) to exit without creating a new ACL
```

Figure 4-14. The USERS Screen

The options that appear on the USERS screen are summarized next:

USER1 = -- Enter the ID of the user to whom you want to grant access. You can list a maximum of 15 users on this screen.

LEVEL1 = -- Enter the access level for USER1. The valid access level options are W for Write, R for Read, and E for Execute.

MORE = -- Enter YES if you want to specify more than 15 individual users; otherwise, enter NO. If you enter YES, the same screen is displayed again so that you can enter 15 more IDs and access levels. You can enter a maximum of 45 individual users in this way. If you enter NO, the GROUPS screen is displayed so that you can enter group IDs and access levels.

The PF key function available from the USERS screen is summarized next:

PF Key	Description
ENTER	Validate entries and Continue -- Continues to another screen (USERS or GROUPS) for specifying the access list. The USERS screen is displayed again if you specified YES for the MORE option. Otherwise, the GROUPS screen is displayed.

After you have entered as many individual user IDs as you want (for a maximum of 45), enter NO for the MORE option. The GROUPS screen (Figure 4-15) is then displayed.

```
Wang VS GETPARM v 7                          Parameter Reference Name: GROUPS
                                                        Message Id: 0100
                                                      Component: GETACL
                        Information Required by PMMLOGON

*** Program File Access Control List ***

Specify Group Id(s) and Access Level required to define the Access List :

GROUP1  = ********  LEVEL1  = *   GROUP2  = ********  LEVEL2  = *
GROUP3  = ********  LEVEL3  = *   GROUP4  = ********  LEVEL4  = *
GROUP5  = ********  LEVEL5  = *   GROUP6  = ********  LEVEL6  = *
GROUP7  = ********  LEVEL7  = *   GROUP8  = ********  LEVEL8  = *
GROUP9  = ********  LEVEL9  = *   GROUP10 = ********  LEVEL10 = *
GROUP11 = ********  LEVEL11 = *   GROUP12 = ********  LEVEL12 = *
GROUP13 = ********  LEVEL13 = *   GROUP14 = ********  LEVEL14 = *
GROUP15 = ********  LEVEL15 = *

                                                      MORE    = NO*
          Press : (ENTER) Validate Entries and Continue Processing
                  (1)  to re-start ACL entry
                  (16) to exit without creating a new ACL
```

Figure 4-15. The GROUPS Screen

The options that appear on the GROUPS screen are summarized next:

GROUP1 = -- Enter the ID of the group to whom you want to grant access. You can enter a maximum of 15 groups on this screen.

LEVEL1 = -- Enter the access level for GROUP1. The valid access level options are W for Write, R for Read, and E for Execute.

MORE = -- Enter YES if you want to specify more than 15 groups; otherwise, enter NO. If you enter YES, the same screen is displayed again so that you can enter 15 more IDs and access levels. You can enter a maximum of 30 groups.

The PF key function available from the GROUPS screen is summarized next:

PF Key	Description
ENTER	Validate entries and Continue Processing -- If you specify YES to the MORE option, the GROUPS screen is displayed again so that you can specify more group IDs. If you specify NO to the MORE option, the access list is assumed to be complete, and Linker processing continues.

After you have entered as many groups as you want (for a maximum of 30), enter NO for the MORE option. Linker processing continues, based on the options you selected and the contents of your link. Other screens may be displayed. For example, if you requested a link map on the OUTPUT screen, the PRINT screen for the link map is displayed next.

4.4.2 Specifying Link Map Print Options

The Linker displays the PRINT screen (Figure 4-16) if you specified YES for the MAP option on the OPTIONS screen or the OUTPUT screen. The PRINT screen enables you to specify the volume, library, and file names of the print file for the link map, and provides options for printing specific portions of the map. To choose the options that correspond to the portions of the link map that you want to print, enter YES or NO.

If you have specified a default access list for your print files (by the Set Usage Constants option of the Command Processor menu), it is assigned to the link map print file. Otherwise, no access list is assigned. The access list identifies the users who can access your file and the type of access granted to them. For information about creating default access lists, refer to the *VS System User's Introduction*.

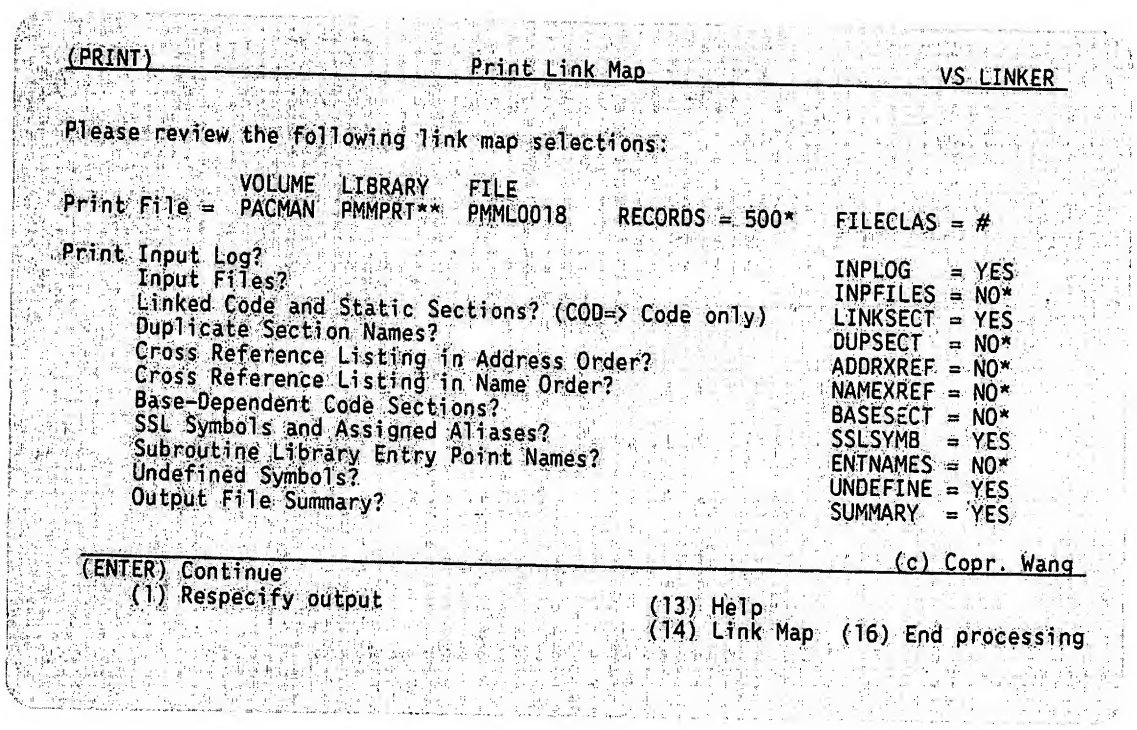

Figure 4-16. The PRINT Screen

The options that appear on the PRINT screen are summarized as follows:

Print file = -- Enter the volume, library, and file names of the print file for the link map.

RECORDS = -- Enter the estimated size of the print file. The default is 500 records.

FILECLAS = -- Enter the file protection class desired. Valid entries are: blank, #, $, @, A-Z. For detailed descriptions of each file class, refer to the *VS System User's Introduction*.

INPLOG = -- Enter YES to print the input log; otherwise, enter NO. The input log contains all Linker options and input specifications.

INPFILES = -- Enter YES to print a list of the input files; otherwise, enter NO. The list also includes the subroutine library files that were included, and, if sections with the same name were specified, the list also shows which sections were selected for the link and which sections were omitted. The input files and the included subroutine library files are listed in the order in which they were specified.

LINKSECT = -- Enter YES to print a list of the linked code sections and static sections. Enter COD to print a list of the linked code sections only; enter NO to omit both listings. Each list organizes the sections in the order in which they were linked, and includes the base address, length, entry point name, translator name, version number, and symbolic flag.

DUPSECT = -- Enter YES to print a list of duplicate section names; otherwise, enter NO.

ADDRXREF = -- Enter YES to print a cross-reference listing arranged in memory location order; otherwise, enter NO. The cross-reference listing includes the location address, the section name and type, the length of the address constant, the direction of relocation, and the symbols referenced by the address constant.

NAMEXREF = -- Enter YES to print a cross-reference listing arranged in symbol name order; otherwise, enter NO. The cross-reference listing includes the name and address of the symbol, the section name and type, and the location of the references.

BASESECT = -- Enter YES to print a list of linked code sections that are base-dependent when they are loaded at runtime; otherwise, enter NO. If you are creating a shared subroutine library (SSL), the default is set to YES. It is important to determine whether any code sections in the output file are base-dependent if you are creating an SSL, because base-dependent code sections cannot be used in an SSL.

SSLSYMB = -- Enter YES to print a list of shared subroutine library symbols with associated SSL aliases, and a count of the number of distinct SSL aliases associated with each symbol; otherwise, enter NO.

ENTNAMES = -- Enter YES to print a list of subroutine library entry point names; otherwise, enter NO.

UNDEFINE = -- Enter YES to print a list of undefined (user-specified) symbols; otherwise, enter NO.

SUMMARY = -- Enter YES to print an output file summary; otherwise, enter NO. The output file summary includes the following information:

- Total lengths of the linked sections (static and code)
- The program entry point name and offset
- The program name
- The program release date
- The version number
- The output file volume, library, and file names
- The number of output records

The functions available from the PRINT screen are summarized as follows:

PF Key	Description
ENTER	Continue -- Accepts the link map print options and continues Linker processing.
1	Respecify output -- Returns to the OUTPUT screen, enabling you to reenter output parameters. The previous output parameters appear on the OUTPUT screen until you change them.
13	Help -- Displays Help text for the PRINT screen. This function is described in Section 1.5.1.
14	Link Map -- Enables you to display portions of the current developmental state of the link map. This function is described in Section 4.5.1.
16	End processing -- Enables you to terminate processing and exit the Linker. This function is described in Section 4.5.3.

To specify the link map print options, enter YES or NO for each print option displayed on the PRINT screen, then press ENTER. The Linker accepts the print options and proceeds to the next screen according to the chosen Linker options. When the link is complete, the print file is entered on the print queue.

4.4.3 Reordering Program Sections

The Linker displays the REORDER screen (Figure 4-17) if you specified YES for the REORDER option on the OUTPUT screen (Figure 4-13), and if the output file contains more than one code section or static section. The REORDER screen enables you to change the sequence of the code and static sections in the output file.

The REORDER screen lists the section names along with their ordinal positions in the output file. The code sections and static sections are in separate lists. First the code sections are displayed for reordering, then the static sections.

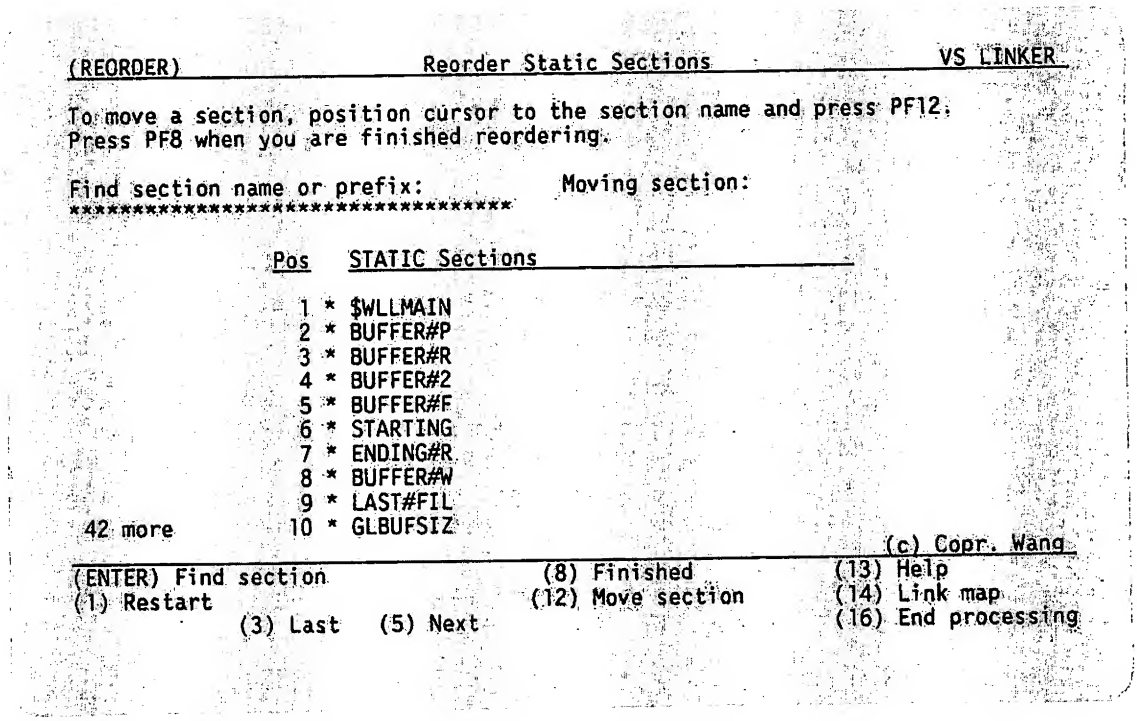

Figure 4-17. The REORDER Screen

The functions available from the REORDER screen are summarized in the following list. Note that First, Last, Prev, and Next appear on the REORDER screen only when there are more than ten sections.

PF Key	Description
ENTER	Find section -- Searches for the first instance of the section name or prefix that you enter in the Find section name or prefix field, and positions the cursor at that location. Press ENTER again to find the next matching section name or prefix, if needed. This function is described in further detail in this section.
1	Restart -- Enables you to erase all input entered thus far and start the link again. This function is described in Section 4.5.2. Cancel -- While you are moving a section (after you have pressed PF12, Move Section) Restart changes to Cancel. Then, PF1 cancels the reordering of the current section and enables you to select another section to reorder.
2	First -- Displays the first ten positions and corresponding sections.

PF Key	Description
3	Last -- Displays the last ten or fewer positions and corresponding sections.
4	Prev -- Displays the previous ten positions and corresponding sections.
5	Next -- Displays the next ten or fewer positions and corresponding sections.
8	Finished -- Ends reordering of the displayed section type. If the type was code sections, the Linker then displays the static sections for reordering. If the type was static sections, the Linker proceeds to the next screen according to the chosen Linker options.
12	Move Section -- Enables you to reorder a section by performing the following steps: 1. Position the cursor at the pseudoblank that precedes the section to be reordered and press PF12. PF8 (Finished) and PF12 (Move section) disappear, PF12 (Move after) and PF↑12 (Move before) appear, and the name of the section being moved appears in the Moving Section field. 2. Position the cursor at the location where you want to move the specified section (the target section). You can use the scrolling keys to scroll the list, or the Find function (enter a name and press ENTER) to move to the target section. 3. Press PF12 to move directly after the target section, or PF↑12 to move directly before the target section. After the Linker performs the task, PF12 (Move after) and PF↑12 (Move before) disappear, and PF8 (Finished) and PF12 (Move section) reappear. You can repeat the above process as many times as necessary, until the required order has been achieved. Then, press PF8 to end reordering of the displayed section type. The reorder function is described in further detail in this section.
13	Help -- Displays Help text for the REORDER screen. This function is described in Section 1.5.1.
14	Link Map -- Enables you to display portions of the current developmental state of the link map. This function is described in Section 4.5.1.

PF Key	Description
16	End processing -- Enables you to terminate processing and exit the Linker. This function is described in Section 4.5.3.

Finding a Program Section

Instead of scrolling through a lengthy list of sections, you can instruct the Linker to find a specified prefix of a section name or the section itself. To do this, enter a prefix of the name or the full name in the Find section name or prefix field, and press ENTER. The Linker then searches the list for the prefix or name entered, and positions the cursor at the location where the first prefix or section name appears. If you press ENTER again from that location, the Linker finds the next occurrence of the prefix or section name. This process is repeated each time you press ENTER.

If you move the cursor to any location outside of the list and press ENTER, the Linker searches for the entered section name or prefix from the beginning of the list.

Note that using a prefix does not ensure that the Linker will find the desired section the first time, since other sections may have a similar name with the same prefix. However, using a prefix with sufficient uniqueness can help you find the section that you want quickly, without having to enter the full name. This is particularly useful when you do not know the exact name of the section, or when the section name is long.

How to Reorder a Program Section

The REORDER screen first displays the code sections of the output file for reordering. To reorder a code section, position the cursor at the pseudoblank that precedes the section name to be reordered, and press PF12. The Linker then alters the PF key selection as follows:

- PF1, Restart, is set to Cancel.
- PF8 (Finished) and PF12 (Move section) are removed.
- PF12 (Move after) and PF12 ((PF28) Move before) are entered.

In addition, the Linker inserts the name of the section being moved in the Moving section field.

Position the cursor at the pseudoblank next to the section name (the target section) that will either precede or follow the section being reordered. You can use the scrolling keys to scroll the list, or the Find function (enter a prefix or name and press ENTER) to move to the target section.

Then, press either PF12 or PF↑12 (PF28). If you press PF12, the Linker moves the section being reordered to the location directly *after* the target section that you indicated with the cursor. If you press PF↑12 (PF28), the Linker moves the section being reordered to the location directly *before* the target section that you indicated with the cursor. After the Linker performs the task, the original PF key selection is restored. You can now proceed to reorder other sections.

When you have reordered all the desired code sections, press PF8. The Linker then displays the static sections, which you can reorder as described for the code sections.

When you have reordered all the desired static sections, press PF8 again to indicate that all reordering is complete. The Linker proceeds to the next screen according to the chosen Linker options.

4.4.4 Assigning Aliases for Shared Subroutine Library Symbols

The Linker displays the SSLALIAS screen (Figure 4-18), if you specified YES for the ALIAS option on the OUTPUT screen (Figure 4-13). You can also select this option on the MOREOPT screen (Figure 4-3), but the setting on the OUTPUT screen overrides what you selected on the MOREOPT screen. The SSLALIAS screen lists all undefined symbols with the corresponding SSL alias (if present) of each symbol. You can accept or modify the existing SSL alias assignments and make additional assignments.

The SSLALIAS screen differs from the ALIASES screen (Figure 4-12) in that you make the assignments explicitly for each symbol. This makes it possible to refer to specific subroutines from alternate libraries. Another important difference is that the Linker does not perform the verification and searching process that it does on the ALIASES screen. This allows you to specify aliases for shared subroutine libraries that exist on other systems.

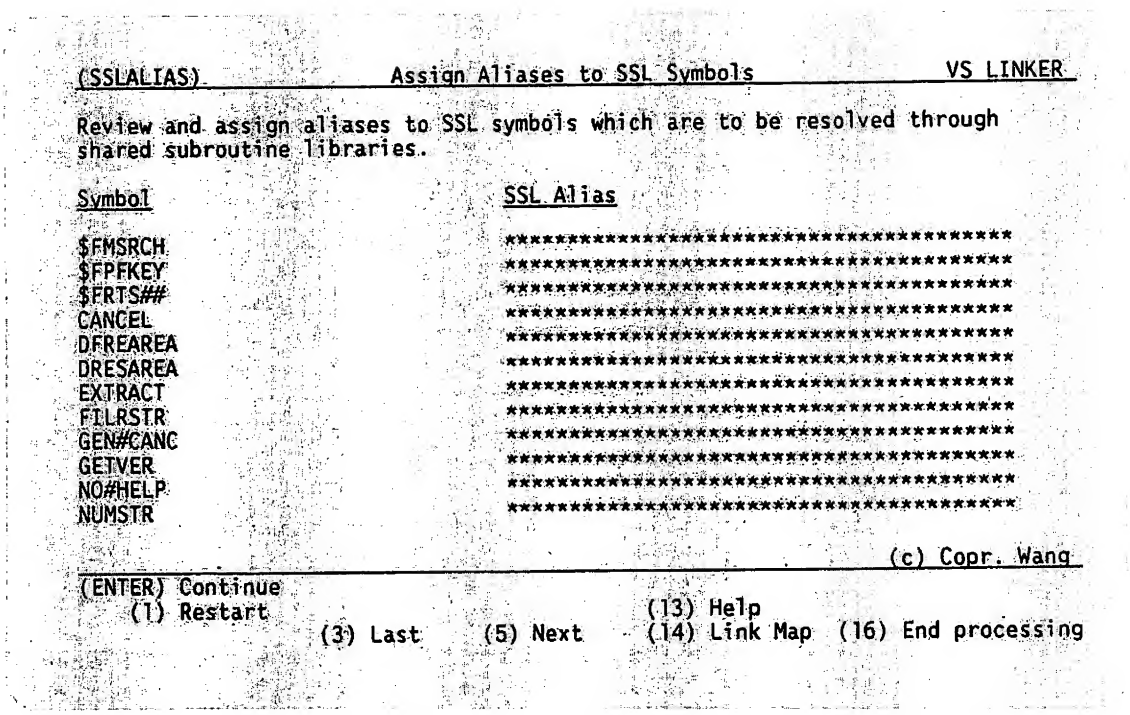

Figure 4-18. The SSLALIAS Screen

The functions available from the SSLALIAS screen are summarized in the following list. Note that First, Last, Prev, and Next appear on the SSLALIAS screen only when there are more than 12 symbols.

PF Key	Description
ENTER	Continue -- Accepts changes in assignment of shared subroutine library aliases to the corresponding symbols.
1	Restart -- Enables you to erase all input entered thus far and start the link again. This function is described in Section 4.5.2.
2	First -- Displays the first 12 symbols and corresponding aliases.
3	Last -- Displays the last 12 or fewer symbols and corresponding aliases.

PF Key	Description
4	Prev -- Displays the previous 12 symbols and corresponding aliases.
5	Next -- Displays the next 12 or fewer symbols and corresponding aliases.
13	Help -- Displays Help text for the SSLALIAS screen. This function is described in Section 1.5.1.
14	Link Map -- Enables you to display portions of the current developmental state of the link map. This function is described in Section 4.5.1.
16	End processing -- Enables you to terminate processing and exit the Linker. This function is described in Section 4.5.3.

To assign aliases for shared subroutine library symbols, examine the list of symbols and aliases and make any desired changes. When you have completed all changes, press ENTER. The Linker proceeds to the next screen according to the chosen Linker options.

4.4.5 Managing the Entry Point Reference (EPR) Table

A subroutine library entry point name is either a section name, or a name defined within a section. An entry point reference (EPR) table contains all subroutine library entry point names that can be used to resolve external references.

If you specified YES for either the STATICSL option or the SHAREDSL option on the OPTIONS screen (Figure 4-2), the Linker automatically builds an EPR table. The Linker collects the section and entry point names for all included code and static sections. After removing duplicate names, the Linker sorts the list in alphabetical order, and places the information in the EPR table. The EPR table is located in the Module block of the output file.

The entry point names in a shared subroutine library can be examined using the SSL utility. For more information, refer to the *VS System Administrator's Reference*.

Selecting Entry Point Names for the EPR Table

The Linker includes all entry point names in the EPR table unless you specify YES for the ENTNAMES option on the OPTIONS screen (Figure 4-2) to select a subset. The Linker then displays the ENTRY screen Figure 4-19) for you to review and select names. If you are creating a static subroutine library, the ENTRY screen lists the entry point names for all included code and static sections. If you are creating a shared subroutine library, only the entry point names for the code sections are listed.

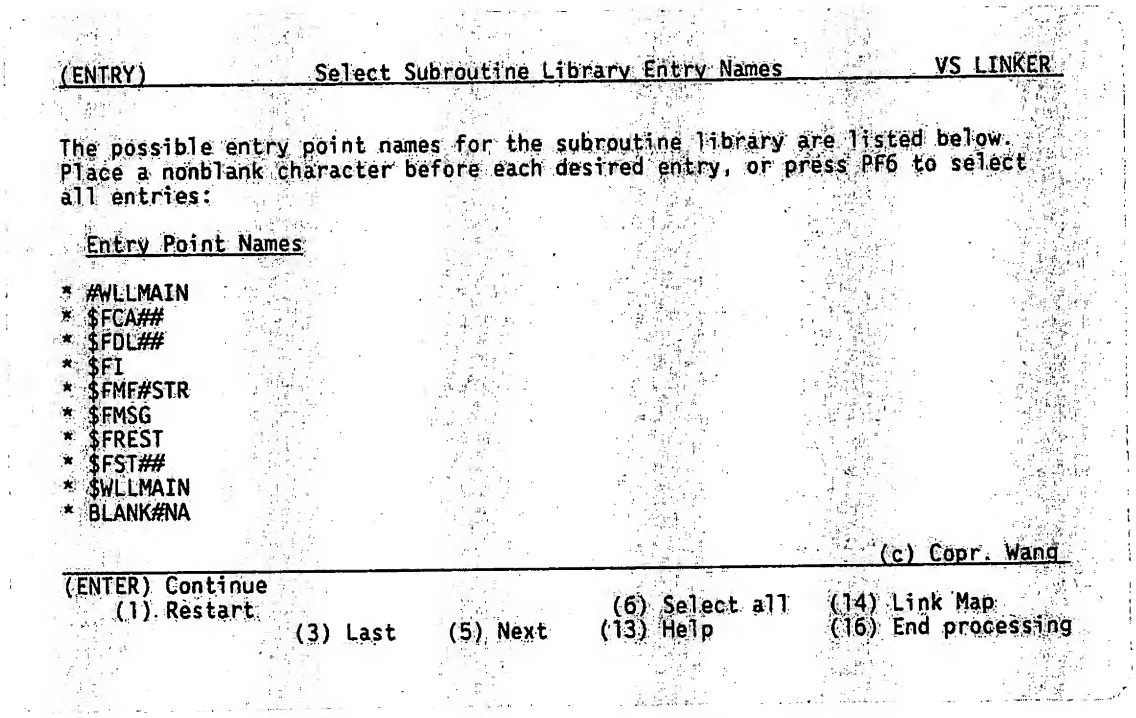

Figure 4-19. The ENTRY Screen

The options that appear on the ENTRY screen are summarized in the following list. First, Last, Prev, and Next appear on the ENTRY screen only when there are more than ten entry point names.

Entry Point Names -- All entry point names are displayed, ten at a time. To select one, place a nonblank character at the pseudoblank preceding the name. To select them all, press PF6.

When the subroutine library file (static or SSL) is used in a subsequent link, only the entry point names that you included in the EPR table are reviewed by the Linker for resolving symbols.

The functions available from the ENTRY screen follow:

PF Key	Description
ENTER	Continue -- Confirms the selection of subroutine library entry points and proceeds to the next screen according to the chosen Linker options.
1	Restart -- Enables you to erase all input entered thus far and start the link again. This function is described in Section 4.5.2.
2	First -- Displays the first 10 entry point names.
3	Last -- Displays the last 10 or fewer entry point names.
4	Prev -- Displays the previous 10 entry point names.
5	Next -- Displays the next 10 or fewer entry point names.
6	Select all -- Marks all entry point names with an X and displays the beginning of the list.
13	Help -- Displays Help text for the ENTRY screen. This function is described in Section 1.5.1.
14	Link Map -- Enables you to display portions of the current developmental state of the link map. This function is described in Section 4.5.1.
16	End processing -- Enables you to terminate processing and exit the Linker. This function is described in Section 4.5.3.

To select an entry point name to be included in the output file, mark the pseudoblank preceding the name with a nonblank character.

To select all entry point names, press PF6. The Linker then marks all entry point names with a nonblank character and displays the beginning of the list.

When you have specified all the desired entry point names, press ENTER. The Linker proceeds to the next screen according to the chosen Linker options.

4.5 COMMON FUNCTIONS

The following functions, described in this section, can be performed from most Linker screens:

- Display the link map
- Restart the Linker
- Terminate the Linker

4.5.1 Displaying the Link Map

The Linker displays the VIEWMAP screen (Figure 4-20) if you press PF14 from any screen that provides this function. The VIEWMAP screen lists functions for viewing specified portions of the link map. When you make a selection, the Linker generates that portion of the link map for examination, and then automatically deletes it when you return to the VIEWMAP screen. Note that the status of the MAP option on the OPTIONS screen has no effect on this function.

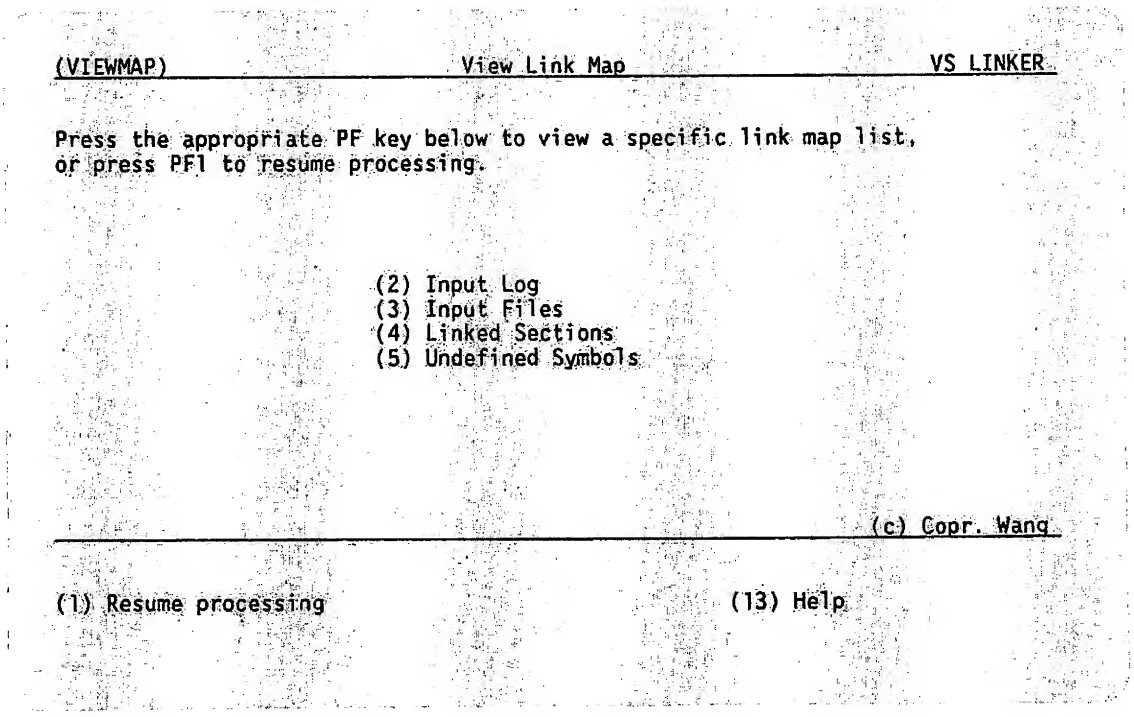

Figure 4-20. The VIEWMAP Screen

The functions available from the VIEWMAP screen follow:

PF Key	Description
1	Resume processing -- Returns to the previous screen.
2	Input Log -- Displays the Input Log portion of the link map.
3	Input Files -- Displays a current list of the input files for the link.
4	Linked Sections -- Displays a current list of the linked sections of the program file.
5	Undefined Symbols -- Displays a current list of the undefined symbols for the link.
13	Help -- Displays Help text for the VIEWMAP screen. This function is described in Section 1.5.1.

4.5.2 Restarting the Linker

The Linker displays the RESTART screen (Figure 4-21) whenever you press PF1 from any Linker screen that provides this function. The RESTART screen prompts you to confirm that you want to cancel all processing up to this point and start again.

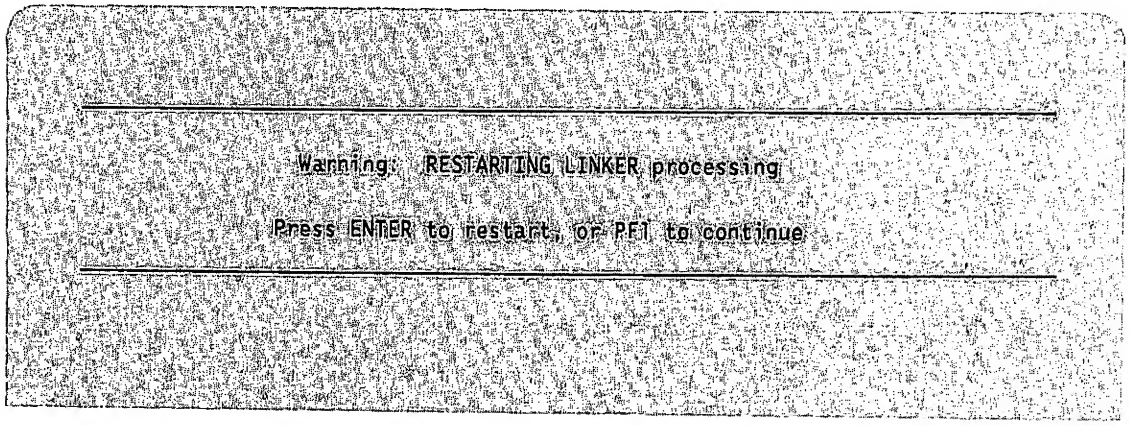

Figure 4-21. The RESTART Screen

The functions available from the RESTART screen follow:

PF Key	Description
ENTER	Restart -- Erases all input entered thus far and displays the OPTIONS screen.
1	Continue -- Returns to the previous screen and enables you to continue Linker processing.

4.5.3 Exiting the Linker

The Linker displays the TERMINATE screen (Figure 4-22) whenever you press PF16 from any Linker screen that provides this function. The TERMINATE screen prompts you to confirm that you want to end processing and exit the Linker.

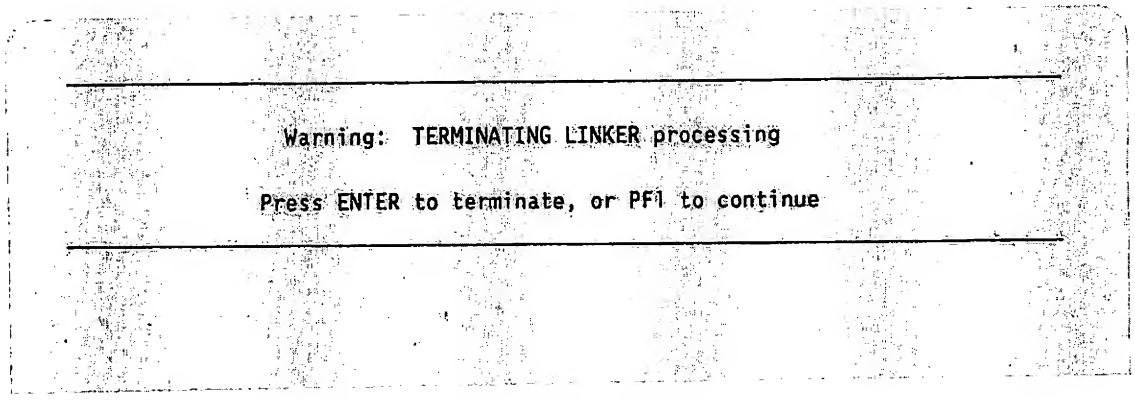

Figure 4-22. The TERMINATE Screen

The functions available from the TERMINATE screen follow:

PF Key	Description
ENTER	Terminate -- Enables you to terminate processing and exit the Linker.
1	Continue -- Returns to the previous screen and resumes Linker processing.

4.6 BUILDING THE OUTPUT FILE

When the Linker has shown all screens according to the chosen Linker options, it builds the output file. After the Linker finishes building the output file, control is returned to the program that invoked the Linker.

CHAPTER 5
LINKER GETPARM INFORMATION FOR VS PROCEDURES

5.1 INTRODUCTION

The VS Procedure language enables you to create procedures to access Linker screens and perform specified tasks. This chapter provides all the Linker information required to write procedures for the new Linker.

By using the VS Procedure language and appropriate GETPARMS, you can automate screen access and task performance to eliminate the display of most or all of the screens. For detailed information on the use of VS Procedure language and GETPARM requests, refer to the *VS Procedure Language Reference*.

The new Linker is procedurally compatible with the previous Linker. In other words, procedures written for the previous Linker will produce the same results with the new Linker, except for procedures that rely upon the INLIB and INVOL usage constants as a default subroutine library (refer to Section 5.7).

The screens that accept data through a VS procedure are presented in alphabetical order. For each screen, the following information is given:

- A brief description of the screen
- The Procedure language format, all prnames and keywords, and any relevant PF keys other than ENTER
- General rules for managing the screen by procedure

5.2 The ALIASES Screen — Prname = ALIASES

The ALIASES screen enables you to resolve undefined symbols through shared subroutine libraries by specifying SSL aliases.

Format

```
[label]    ENTER    ALIASES
                ALIAS1    = ssl-alias
           [,   ALIAS2    = ssl-alias]
           [,   ALIAS3    = ssl-alias]
           [,   ALIAS4    = ssl-alias]
           [,   ALIAS5    = ssl-alias]
           [,   ALIAS6    = ssl-alias]
           [,   ALIAS7    = ssl-alias]
           [,   ALIAS8    = ssl-alias]
           [,   ALIAS9    = ssl-alias]
           [,   ALIAS10   = ssl-alias]
```

General Rules

1. The ENTER ALIASES clause is processed only if there are undefined symbols remaining after all input files and static subroutine libraries have been searched.

2. Only one ENTER ALIASES clause is accepted in a procedure.

3. An SSL alias is a name of 40 characters or less assigned to a shared subroutine library.

4. A maximum of 10 aliases can be specified.

5.3 The ENTRY Screen — Prname = ENTRY

The ENTRY screen displays the entry point names for all included code and static sections. This screen enables you to choose the entry point names to be included in the entry point reference table for a subroutine library file.

Format 1

```
[label]    ENTER    ENTRY
             NAME1    = entry-point-name

           [,    NAME2    = entry-point-name]
          .
          .
          .
           [,    NAME10   = entry-point-name]
```

Format 2

```
[label]    ENTER   ENTRY
```

General Rules

1. ENTER ENTRY clauses are processed only when ENTNAMES and STATICSL or SHAREDSL = YES in the OPTIONS parameter list.

2. More than one ENTER ENTRY clause can be used to specify entry point names.

3. A Format 2 ENTER ENTRY clause is used to terminate a sequence of ENTRY parameter inputs.

4. The procedure is checked first for any ENTRY parameters. All parameter lists are processed until a Format 2 ENTER clause is encountered.

   If a Format 2 ENTER ENTRY clause is not specified, the following occurs:

   a. When the procedure is running in the foreground, the ENTRY screen is displayed at the workstation, with data extracted from the procedure listed on the screen.

   b. When the procedure is running in the background, the data extracted from the procedure is assumed to be complete.

5. The ENTRY screen is displayed if the following conditions are true:

   - The procedure is running in the foreground.
   - ENTNAMES and STATICSL or SHAREDSL = YES on the OPTIONS screen.
   - No Format 2 statement is included.

5.4 The EXCLUDE Screen — Prname = EXCLUDE

The EXCLUDE screen enables you to specify sections to be excluded from the output file.

Format

```
[label]    ENTER    EXCLUDE
          VOLUME   = VS-volume-name,
          LIBRARY  = VS-library-name,
          FILE     = VS-file-name,
          SECT1    = section-name
           [,    SECT2    = section-name]
           [,    SECT3    = section-name]
           [,    SECT4    = section-name]
           [,    SECT5    = section-name]
           [,    SECT6    = section-name]
           [,    SECT7    = section-name]
           [,    SECT8    = section-name]
           [,    SECT9    = section-name]
           [,    SECT10   = section-name]
           [,    BLNKSECT = YES-or-NO]
```

General Rules

1. The SECT, VOLUME, LIBRARY, and FILE names are required.

2. All sections specified in an ENTER EXCLUDE clause must belong to the same input file.

3. More than one ENTER EXCLUDE clause can be used to specify all excluded sections for the same file.

4. For each input file, the following rules apply when specifying excluded sections:

   a. The procedure is checked first for any ENTER EXCLUDE clauses with a matching file name specification (volume, library, and file). When such a clause exists, the sections named in the ENTER EXCLUDE clause are marked as excluded.

   b. If the procedure is running in the foreground, and the EXCLUDE option for the input file is YES, the EXCLUDE screen is displayed. The EXCLUDE screen contains all the sections in the input file except those already excluded through the procedure. The value of the EXCLUDE option is derived from either the INPUT parameter list or the INPUT screen.

   c. If BLNKSECT = YES, the blank section is excluded. The default is NO.

5.5 The FILELIST Screen — Prname = FILELIST

The FILELIST screen enables you to specify individual files to link from the library that you identified on the INPUT screen. From a procedure, you can only include all of the files; you cannot specify individual files.

Format

```
[label]    ENTER    FILELIST

           PF key:  6 = Include all
```

General Rules

1. PF6 is required with the ENTER FILELIST clause.

2. Only one ENTER FILELIST clause can be specified for each INPUT library specification.

5.6 The INPUT Screen — Prname = INPUT

The INPUT screen enables you to specify one or more input files to link. You can specify an unlimited number of input files: either individual files, or a library of files from which you can select files on the FILELIST screen.

Format 1

```
[label]    ENTER     INPUT
          [  VOLUME    = VS-volume-name,]
          [  LIBRARY   = VS-library-name,]
          [  FILE      = VS-file-name]
          [, SYMBOLIC  = YES-or-NO]
          [, EXCLUDE   = YES-or-NO]
```

Format 2

```
[label]    ENTER     INPUT
```

General Rules

1. If the procedure is run in the background, there must be at least one input file or library specification.

2. The LIBRARY and FILE fields are required to specify a file. The system uses the user input volume as the default for the VOLUME field. To specify a library, omit the FILE field. If EXCLUDE is not specified, a default value of NO is assigned. If SYMBOLIC is not specified, the value that is consistent with the global SYMBOLIC option is assigned.

3. If a library is specified when the procedure is running in the background, there must be a corresponding ENTER FILELIST 6 clause to select the files; otherwise, the Linker will be canceled when it attempts to list the files on the workstation.

4. When the procedure is running in the foreground and EXCLUDE = YES, the EXCLUDE screen for the input file is displayed. Otherwise, the EXCLUDE field is ignored.

5. A Format 2 INPUT parameter is used to terminate a sequence of INPUT parameters. If a Format 2 INPUT parameter is not specified, the following results occur:

   a. When the procedure is running in the foreground, an INPUT screen is displayed at the workstation.

   b. When the procedure is running in the background, the Linker will be canceled in an attempt to access more data at the workstation.

5.7 The LIBRARY Screen — Prname = LIBRARY

The LIBRARY screen enables you to enter static subroutine libraries to resolve undefined symbols.

Format 1

```
[label]    ENTER     LIBRARY
          VOLUME1    = sublib-volume-name,
          LIBRARY1   = sublib-library-name
           [,    FILE1      = sublib-file-name]

           [,    VOLUME2    = sublib-volume-name,
          LIBRARY2   = sublib-library-name
           [,    FILE2      = sublib-file-name]]

              .
          .
          .

           [,    VOLUME8    = sublib-volume-name,
          LIBRARY8   = sublib-library-name
           [,    FILE8      = sublib-file-name]]
```

Format 2

```
[label]    ENTER     LIBRARY
```

General Rules

1. All static subroutine libraries must be specified using an ENTER LIBRARY clause. A default subroutine library is not supplied from the INLIB and INVOL usage constants as in the previous Linker.

2. Processing of ENTER LIBRARY clauses terminates when the eighth subroutine library specification in an ENTER LIBRARY clause is blank.

3. More than one ENTER LIBRARY clause can be used to specify static subroutine libraries.

4. A Format 2 ENTER LIBRARY clause is required only if one of the following conditions is true:

   a. Undefined symbols exist, but there are no static subroutines to be linked (all undefined symbols are to be resolved through shared subroutine libraries).

   b. The eighth subroutine library specification in the preceding ENTER LIBRARY clause is not blank.

5.8 The OPTIONS Screen — Prname = OPTIONS

Note: *The GETPARM list for this screen includes the options that appear on the MOREOPT screen.*

The OPTIONS screen enables you to specify the most commonly used Linker options. Note that you also specify keywords from the MOREOPT screen using the ENTER OPTIONS clause.

Format

```
[label]    ENTER    OPTIONS
       [, MAP       = YES-or-NO]
       [, SYMBOLIC  = YES-or-NO]
       [, LINKAGE   = YES-or-NO]
       [, STATICSL  = YES-or-NO]
       [, SHAREDSL  = YES-or-NO]
       [, ENTNAMES  = YES-or-NO]
       [, ALIAS     = YES-or-NO]
       [, REORDER   = YES-or-NO]
       [, RESOLVE   = YES-or-NO]
       [, DUPSECT   = YES-or-NO]
       [, INPROGRS  = YES-or-NO]
       [, OBJFORM   = 0-thru-1]
       [, WARNINGS  = YES-or-NO]
       [, DATESEL   = YES-or-NO]
```

General Rules

1. Only one ENTER OPTIONS clause is accepted in a procedure.

2. The ENTER OPTIONS clause is typically placed after the RUN statement.

3. The INLIB and INVOL usage constants are not supplied as a default subroutine library as in the previous Linker. All required subroutine libraries must be explicitly specified with an ENTER LIBRARY clause (refer to Section 5.7).

4. Keywords of the previous Linker are supported as follows:

   VOLUME, LIBRARY -- Becomes the first subroutine library on the LIBRARY screen.

   MORE -- If YES, the LIBRARY screen is displayed; if NO, the LIBRARY screen is not displayed.

   SYMB -- Equivalent to the SYMBOLIC option.

EXSEC -- Equivalent to the INPFILES option on the PRINT screen.

XREF -- Equivalent to the ADDRXREF option on the PRINT screen.

5.9 The OUTPUT Screen — Prname = OUTPUT

The OUTPUT screen shows a list of output file attributes. From that list, you can specify the attributes of the output file.

Format

```
[label]    ENTER    OUTPUT
          [  VOLUME   = VS-volume-name,]
          [  LIBRARY  = VS-library-name,]
          [  FILE     = VS-file-name]
          [, REPLACE  = YES or NO]
          [, REORDER  = YES or NO]
          [, ALIAS    = YES or NO]
          [, MAP      = YES or NO]
          [, FILECLAS = VS-file-class]
          [, ACLIST   = YES or NO]
          [, RETAIN   = #-of-days-in-integer]
          [, PROGBASE = program-base-address-in-hex]
          [, PROGNAME = program-name]
          [, ENTRY    = program-entry-name]
          [, ENOFFSET = entry-point-name-offset-in-hex]
          [, VERSION1 = 2-digit-no]
          [, VERSION2 = 2-digit-no]
          [, VERSION3 = 2-digit-no]
          [, RELMONTH = 1-thru-12]
          [, RELDAY   = 1-thru-31]
          [, RELYEAR  = 00-thru-99]
```

General Rules

1. The output parameter list is required for background procedures.

2. The FILE field is required. Default values are used for the output library and volume names if they are not specified.

3. A FILE specification of '##' is acceptable to create a temporary file. The actual file, library, and volume names are returned to the procedure.

4. The PROGBASE and ENOFFSET fields can have a maximum of six hexadecimal digits.

5. The PROGNAME and ENTRY fields can have a maximum of 40 alphanumeric characters.

6. VERSION1, VERSION2, and VERSION3 field values are used to form the following version number: VERSION1.VERSION2.VERSION3

7. RELMONTH, RELDAY, and RELYEAR field values are used to form the following date: RELMONTH / RELDAY / RELYEAR

8. The version number and release date are placed in the Prolog block. The release date is converted to a Julian date and stored in packed decimal in the Prolog block.

9. Only one ENTER OUTPUT clause is accepted in a procedure.

5.10 The OVERRIDE Screen — Prname = OVERRIDE

Note: *The OVERRIDE screen replaces the DUPSECT screen when you are using a procedure to run the LINKER. The DUPSECT screen is designed for interactive use only.*

The OVERRIDE screen enables you to specify which sections to link when there are duplicate section names.

Format

```
[label]    ENTER    OVERRIDE
          SECT1         = section-name,
          VOLUME1       = VS-volume-name,
          LIBRARY1      = VS-library-name,
          FILE1         = VS-file-name

           [,    SECT2         = section-name,
          VOLUME2       = VS-volume-name,
          LIBRARY2      = VS-library-name,
          FILE2         = VS-file-name]

          .
          .
          .

           [,    SECT8         = section-name,
          VOLUME8       = VS-volume-name,
          LIBRARY8      = VS-library-name,
          FILE8         = VS-file-name]

           [,    BLNKVOL       = VS-volume-name,
                 BLNKLIB       = VS-library-name,
                 BLNKFILE      = VS-file-name]
```

General Rules

1. The SECT, VOLUME, LIBRARY, and FILE fields are required in each set of parameters.

2. More than one ENTER OVERRIDE clause can be used to specify all overriding sections.

3. For each duplicate section, the following process is used to enable you to specify an overriding section:

   a. The procedure is checked first for a set of OVERRIDE parameters (a set includes the SECT, VOLUME, LIBRARY, and FILE names) with a matching section-name entered in the SECT field. If such a set is found, the section from the file specified in the set is selected as the overriding section. If none exist, and DUPSECT = YES on the OPTIONS screen when running in the foreground, the DUPSECT screen is displayed; otherwise, the default selection rules are used.

   b. By specifying BLNKVOL, BLNKLIB, and BLNKFILE, you can select the blank section when multiple copies from multiple files are supplied.

5.11 The PRINT Screen — Prname = PRINT

The PRINT screen enables you to enter the name and location of the print file for the link map, and allows you to select which portions of the link map are to be printed. In a procedure, if MAP = NO, the ENTER PRINT clause enables you to specify the name and location of the print file for warning messages.

Format

```
[label]    ENTER    PRINT
          [  VOLUME    = VS-volume-name,]
          [  LIBRARY   = VS-library-name,]
          [  FILE      = VS-file-name]
          [, INPLOG    = YES-or-NO]
          [, INPFILES  = YES-or-NO]
          [, LINKSECT  = YES or COD or NO]
          [, DUPSECT   = YES-or-NO]
          [, BASESECT  = YES-or-NO]
          [, ADDRXREF  = YES-or-NO]
          [, NAMEXREF  = YES-or-NO]
          [, SSLSYMB   = YES-or-NO]
          [, ENTNAMES  = YES-or-NO]
          [, UNDEFINE  = YES-or-NO]
          [, SUMMARY   = YES-or-NO]
          [, FILECLAS  = blank, #, $, @, or A-Z]
```

General Rules

1. If MAP = YES, the file, library, and volume names specify the name and location of the link map file. If MAP = NO, the file, library, and volume names specify the name and location of the file for printing only the warning messages; the remainder of the parameter list is ignored.

2. If the VOLUME, LIBRARY, and FILE fields are not specified, the print file is created with a system-generated file name, in the user's default spool library and volume.

3. A file specification of '##' is acceptable for creating a temporary file. The actual file, library, and volume names of the created file are returned to the procedure.

4. LINKSECT = YES specifies that both code and static sections are to be listed in the link map. LINKSECT = COD specifies that only the code sections are to be listed. LINKSECT = NO specifies that neither the code nor the static sections are to be listed.

5.12 The REORDER Screen — Prname = REORDER

The REORDER screen enables you to change the sequence of the code and static sections in the output file.

Format 1

```
[label]    ENTER    REORDER
          TYPE        = CODE-OR-STATIC
          [, SECT1    = section-name]
          [, SECT2    = section-name]
             .
             .
             .
          [, SECT15   = section-name]
          [, OTHERS   = section-name]
          [, BLKSCTAF = section-name]
          [, BLKSCTBF = section-name]
```

Format 2

```
[label]    ENTER    REORDER
```

General Rules

1. The keyword TYPE is required, and specifies whether the list is to be applied to code sections or static sections.

2. The keyword OTHERS can be used to place the remaining unspecified sections after the section that OTHERS names. The sections are entered in their input order.

3. ENTER REORDER clauses are processed only when REORDER = YES in either the OPTIONS parameter list or the OUTPUT parameter list.

4. More than one ENTER REORDER clause can be used to specify all sections for reordering for the same file.

5. A Format 2 ENTER REORDER clause that contains no parameters is used to terminate a sequence of REORDER parameter inputs. If a Format 2 ENTER REORDER clause is not specified, the following results occur:

   a. When the procedure is running in the foreground, the REORDER screen is displayed at the workstation. The REORDER screen lists the sections in the order indicated by the parameter list. You can further manipulate the order of the sections.

   b. When the procedure is running in the background, the end of the REORDER list is assumed.

6. The BLKSCTAF keyword is used to designate a section after which the blank section will appear.

7. The BLKSCTBF keyword is used to designate a section before which the blank section will appear.

5.13 The RESOLVE Screen — Prname = RESOLVE

The RESOLVE screen enables you to specify the action you want to take if undefined symbols exist after all input has been processed. The RESOLVE prname is ignored if the procedure is running in background mode.

Format

```
[label]    ENTER    RESOLVE      [PFkey]

          PF keys: 1 = Add more input files
                   8 = Verify and add aliases
                   9 = Display undefined symbols
                  13 = Help
                  14 = Link map
                  16 = End processing
```

General Rule

1. The ENTER RESOLVE clause is processed only when RESOLVE = YES in the OPTIONS parameter list and the procedure is running in foreground mode.

2. If the PF key is omitted, the Linker bypasses the RESOLVE screen and proceeds to create the output file. The same effect is achieved when RESOLVE = NO in the OPTIONS parameter list.

3. PF13 and PF14 are not permitted if the procedure is running in the background.

5.14 The SSLALIAS Screen — Prname = SSLALIAS

The SSLALIAS screen displays all undefined symbols with the corresponding SSL alias (if present) of each symbol and allows you to change the current SSL assignments and to make additional assignments.

Format

```
[label]    ENTER    SSLALIAS

            [,    NAME1   = symbol-name,
           ALIAS1  = ssl-alias]

            [,    NAME2   = symbol-name,
           ALIAS2  = ssl-alias]

           .
           .
           .

            [,    NAME10  = symbol-name,
           ALIAS10 = ssl-alias]
```

General Rules

1. The symbol name and alias fields are required. Up to 40 characters can be entered for a symbol name or an SSL alias.

2. An SSL alias is a name of 40 characters or less assigned to a shared subroutine library.

3. To display alias assignments for shared subroutine library symbols, the ALIAS option must be set to YES. The ALIAS option can be specified in either the OPTIONS parameter list, or the OUTPUT parameter list.

4. More than one ENTER SSALIAS clause can be used to assign SSL aliases to symbols.

5. The SSLALIAS screen is displayed if all of the following conditions are true:

   - The procedure is running in the foreground.
   - There are no SSLALIAS parameters in the procedure.
   - ALIAS = YES in the OUTPUT parameter list.

APPENDIX A
LINK MAPS

A.1 INTRODUCTION

This appendix gives examples of link maps from two types of links: one from linking a program and the other from linking a shared subroutine library (SSL). The objective is to illustrate the form and content of the information provided in the link maps produced by the Linker. Included with the link maps are the corresponding link procedures.

Information in a link map is organized into sections, where each section starts on a new page with a section heading. There are 11 sections that can be selected for printing on the PRINT screen, plus a section for warning messages. The Linker provides default selections according to the type of link you are performing. You can accept the defaults or make other selections. For more information on link map selections, refer to Section 2.4.1.

All the link map sections are represented in this appendix except for Duplicate Sections and Base-Dependent Code Sections. These two sections are not applicable in the examples given here and, therefore, are discussed in other appendixes (see Appendix B and Appendix C).

A.2 EXAMPLE OF A PROGRAM LINK

This section gives an example of a typical program link. The program being linked consists of three input modules and calls to several subroutines, some of which are contained within a shared subroutine library, and some of which are contained in other subroutine libraries.

The program calls four subroutines for doing workstation I/O: CRTOPEN, CRTCLOSE, CRTREAD, and CRTWRITE. These subroutines are contained in the SSL named @CRTIO. The ENTER ALIASES clause gives the name of the SSL where the Linker will find the subroutine entry point names. The Linker verifies that these entry points exist in the SSL and then assigns the alias to the SSL symbols (refer to Section A.2.8).

The program also calls a subroutine named BELL in the USERSUBS library, and several runtime routines in the @PLIRTM@ library. These subroutines are statically linked in the program. Two subroutines, SETTIME and SETALARM, were intentionally omitted from the link to illustrate the "Undefined Symbols" section of the link map.

A.2.1 VS Procedure for Linking a Program

```
PROCEDURE
RUN LINKER
    ENTER OPTIONS
    ENTER INPUT     FILE=MAIN,      LIBRARY=DRDOBJ,   VOLUME=LANG
    ENTER INPUT     FILE=TIME,      LIBRARY=DRDOBJ,   VOLUME=LANG
    ENTER INPUT     FILE=RINGBELL,  LIBRARY=DRDOBJ,   VOLUME=LANG
    ENTER INPUT
    ENTER LIBRARY   LIBRARY1=USERSUBS, VOLUME1=SYSTEM,
                    LIBRARY2=@PLIRTM@, VOLUME2=SYSTEM
    ENTER ALIASES   ALIAS1=@CRTIO
    ENTER OUTPUT    FILE=CLOCK,     LIBRARY=DRDRUN,   VOLUME=LANG
    ENTER PRINT     FILE=CLOCK,     LIBRARY=DRDPRT,   VOLUME=LANG,
        INPLOG      = YES,
        INPFILES    = YES,
        ADDRXREF    = YES,
        NAMEXREF    = YES,
        SSLSYMB     = YES
```

A.2.2 Program Link Map -- Input Log

```
VS LINKER ( 2.05.00)     12:11     11/01/88                    INPUT LOG -  1

INPUT LOG:

    OPTIONS:
        MAP                                 YES
        SYMBOLIC                            YES
        LINKAGE                             YES
        DATESEL                             NO
        DUPSECT                             NO
        REORDER                             NO
        RESOLVE                             NO
        OBJFORM                             1
        STATIC SUBROUTINE LIBRARY           NO
        SHARED SUBROUTINE LIBRARY           NO
        REVIEW ENTRY POINT NAMES            NO
        REVIEW ALIAS ASSIGNMENTS            NO

    INPUT FILE STRINGS:
        LANG  .DRDOBJ  .MAIN
        LANG  .DRDOBJ  .TIME
        LANG  .DRDOBJ  .RINGBELL

    SUBROUTINE LIBRARY STRINGS:
        SYSTEM.USERSUBS.
        SYSTEM.@PLIRTM@.

    OUTPUT PARAMETERS:
        OUTPUT FILE                         LANG.DRDRUN.CLOCK
        FILE CLASS                          H
        RETAIN                                0 DAY(S)
        PROGRAM BASE ADDRESS                100000
        PROGRAM NAME
        ENTRY POINT NAME                    MAIN
        ENTRY POINT OFFSET                  000000
        VERSION NUMBER                      00.00.00
        RELEASE DATE                        11/01/88

    PRINT LINK MAP:
        LINK MAP FILE                       LANG.DRDPRT.CLOCK
        INPUT LOG                           YES
        INPUT FILES                         YES
        LINKED SECTIONS                     YES
        DUPLICATE SECTIONS                  NO
        XREF BY ADDRESS                     YES
        XREF BY SYMBOL                      YES
        BASE-DEPENDENT SECTIONS             NO
        SSL SYMBOLS AND ALIASES             YES
        SUBROUTINE ENTRY NAMES              NO
        UNDEFINED SYMBOLS                   YES
        OUTPUT FILE SUMMARY                 YES
```

A.2.3 Program Link Map -- Input Files

```
VS LINKER ( 2.05.00)     12:11     11/01/88                    INPUT FILES -  1
INPUT FILES:

FILE = LANG  .DRDOBJ  .MAIN       SYMBOLIC = YES;  INPUT TYPE = USER

    SECTIONS INCLUDED:
        #MAIN
        $MAIN

FILE = LANG  .DRDOBJ  .TIME       SYMBOLIC = YES;  INPUT TYPE = USER

    SECTIONS INCLUDED:
        TIME

FILE = LANG  .DRDOBJ  .RINGBELL   SYMBOLIC = YES;  INPUT TYPE = USER

    SECTIONS INCLUDED:
        #RINGBEL
        $RINGBEL

FILE = SYSTEM.USERSUBS.BELL       SYMBOLIC = YES;  INPUT TYPE = SUBROUTINE LIBRARY

    SECTIONS INCLUDED:
        BELL
        BELLST

FILE = SYSTEM.USERSUBS.DATE       SYMBOLIC = YES;  INPUT TYPE = SUBROUTINE LIBRARY

    SECTIONS INCLUDED:
        DATE

FILE = SYSTEM.@PLIRTM@.WLPALLOC   SYMBOLIC = YES;  INPUT TYPE = SUBROUTINE LIBRARY

    SECTIONS INCLUDED:
        WLPALLOC

FILE = SYSTEM.@PLIRTM@.WLPFREE    SYMBOLIC = YES;  INPUT TYPE = SUBROUTINE LIBRARY

    SECTIONS INCLUDED:
        WLPFREE
```

A.2.4 Program Link Map -- Linked Code Sections

```
VS LINKER ( 2.05.00)     12:11     11/01/88                LINKED SECTIONS -  1

LINKED CODE SECTIONS:

ORIGIN LENGTH  SECTION NAME
       INPUT FILE                  TRANSLATOR   VERSION   DATE      TIME

100040 000130  #MAIN
       LANG  .DRDOBJ  .MAIN        PL           02.01.00  11/01/88  00:00
       *** ENTRY POINTS ***
       OFFSET  ADDRESS   NAME
       000000  <100040>  #MAIN
       000004  <100044>  MAIN

100170 000018  TIME
       LANG  .DRDOBJ  .TIME        AS           01.09.03  09/02/85  00:00
       *** ENTRY POINTS ***
       OFFSET  ADDRESS   NAME
       000000  <100170>  TIME

100188 000090  #RINGBEL
       LANG  .DRDOBJ  .RINGBELL    PL           02.01.00  11/01/88  00:00
       *** ENTRY POINTS ***
       OFFSET  ADDRESS   NAME
       000000  <100188>  #RINGBEL
       000004  <10018C>  RINGBELL

100218 000D10  DATE
       SYSTEM.USERSUBS.DATE        AS           01.07.00  09/17/80  00:00
       *** ENTRY POINTS ***
       OFFSET  ADDRESS   NAME
       000000  <100218>  DATE

100F28 000088  WLPALLOC
       SYSTEM.@PLIRTM@.WLPALLOC    AS           01.09.03  07/19/85  00:00
       *** ENTRY POINTS ***
       OFFSET  ADDRESS   NAME
       000000  <100F28>  WLPALLOC

100FB0 000040  WLPFREE
       SYSTEM.@PLIRTM@.WLPFREE     AS           01.09.03  07/19/85  00:00
       *** ENTRY POINTS ***
       OFFSET  ADDRESS   NAME
       000000  <100FB0>  WLPFREE

100FF0 000220  BELL
       SYSTEM.USERSUBS.BELL        AS           01.07.00  05/16/79  00:00
       *** ENTRY POINTS ***
       OFFSET  ADDRESS   NAME
       000000  <100FF0>  BELL
```

A.2.5 Program Link Map -- Linked Static Sections

```
VS LINKER ( 2.05.00)     12:11     11/01/88                LINKED SECTIONS -  2

LINKED STATIC SECTIONS:

ORIGIN LENGTH  SECTION NAME
        INPUT FILE                TRANSLATOR   VERSION    DATE      TIME

000000 000048  $MAIN
        LANG  .DRDOBJ  .MAIN      PL           02.01.00   11/01/88  00:00
        *** ENTRY POINTS ***
        OFFSET  ADDRESS   NAME
        000000  <000000>  $MAIN

000048 000008  $RINGBEL
        LANG  .DRDOBJ  .RINGBELL  PL           02.01.00   11/01/88  00:00
        *** ENTRY POINTS ***
        OFFSET  ADDRESS   NAME
        000000  <000048>  $RINGBEL

000050 000090  BELLST
        SYSTEM.USERSUBS.BELL      AS           01.07.00   05/16/79  00:00
        *** ENTRY POINTS ***
        OFFSET  ADDRESS   NAME
        000000  <000050>  BELLST
```

A.2.6 Program Link Map -- Cross Reference by Address

```
VS LINKER ( 2.05.00)     12:11     11/01/88                XREF BY ADDRESS -  1

CROSS REFERENCE BY ADDRESS:

    ADDRESS TYPE LEN +/-  REFERS TO SYMBOL     DEFINED AT   IN SECTION   TYPE

CODE SECTION NAME =   #MAIN
    100118  RCON  4   +   $MAIN                 000000       $MAIN        STATIC

CODE SECTION NAME =   #RINGBEL
    1001CC  RCON  4   +   $RINGBEL              000048       $RINGBEL     STATIC

CODE SECTION NAME =   DATE
    100EC0  ACON  4   +   <LOCAL REFERENCE>
    100EC4  ACON  4   +   <LOCAL REFERENCE>
    100EC8  ACON  4   +   <LOCAL REFERENCE>
    100ECC  ACON  4   +   <LOCAL REFERENCE>
    100ED0  ACON  4   +   <LOCAL REFERENCE>
    100ED4  ACON  4   +   <LOCAL REFERENCE>
    100ED8  ACON  4   +   <LOCAL REFERENCE>
    100EDC  ACON  4   +   <LOCAL REFERENCE>
    100EE0  ACON  4   +   <LOCAL REFERENCE>
    100F08  ACON  4   +   <LOCAL REFERENCE>

CODE SECTION NAME =   BELL
    101208  RCON  4   +   BELLST              000050       BELLST       STATIC

STATIC SECTION NAME = $MAIN
    000000  ACON  4   +   MAIN                 100044       #MAIN        CODE
    000004  ACON  4   +   DATE                 100218       DATE         CODE
    000008  ACON  4   +   TIME                 100170       TIME         CODE
    00000C  ACON  4   +   CRTOPEN              UNDEFINED
    000010  ACON  4   +   CRTREAD              UNDEFINED
    000014  ACON  4   +   CRTWRITE             UNDEFINED
    000018  ACON  4   +   CRTCLOSE             UNDEFINED
    00001C  ACON  4   +   SETALARM             UNDEFINED
    000020  ACON  4   +   SETTIME              UNDEFINED
    000024  ACON  4   +   RINGBELL             10018C       #RINGBEL     CODE
    00003C  ACON  4   +   WLPALLOC             100F28       WLPALLOC     CODE
    000040  ACON  4   +   WLPFREE              100FB0       WLPFREE      CODE

STATIC SECTION NAME = $RINGBEL
    000048  ACON  4   +   RINGBELL             10018C       #RINGBEL     CODE
    00004C  ACON  4   +   BELL                 100FF0       BELL         CODE

STATIC SECTION NAME = BELLST
    00006C  ACON  4   +   <LOCAL REFERENCE>
```

A.2.7 Program Link Map -- Cross-Reference by Symbol

```
VS LINKER ( 2.05.00)     12:11     11/01/88                    XREF BY SYMBOL -  1

CROSS REFERENCE BY SYMBOL:

SYMBOL     ADDRESS              TYPE      LEN   +/-   SECTION TYPE   SECTION NAME

#MAIN
           DEF: 100040                                  CODE           #MAIN
           REF: NO REFERENCES

#RINGBEL
           DEF: 100188                                  CODE           #RINGBEL
           REF: NO REFERENCES

$MAIN
           DEF: 000000                                  STATIC         $MAIN
           REF: 100118          RCON      4     +      CODE           #MAIN
                000000          ACON      4     +      STATIC         $MAIN

$RINGBEL
           DEF: 000048                                  STATIC         $RINGBEL
           REF: 1001CC          RCON      4     +      CODE           #RINGBEL
                000048          ACON      4     +      STATIC         $RINGBEL

BELL
           DEF: 100FF0                                  CODE           BELL
           REF: 00004C          ACON      4     +      STATIC         $RINGBEL

BELLST
           DEF: 000050                                  STATIC         BELLST
           REF: 101208          RCON      4     +      CODE           BELL
                000050          ACON      4     +      STATIC         BELLST
                00006C          ACON      4     +      STATIC         BELLST

CRTCLOSE
           DEF: UNDEFINED
           REF: 000018          ACON      4     +      STATIC         $MAIN

CRTOPEN
           DEF: UNDEFINED
           REF: 00000C          ACON      4     +      STATIC         $MAIN

CRTREAD
           DEF: UNDEFINED
           REF: 000010          ACON      4     +      STATIC         $MAIN

CRTWRITE
           DEF: UNDEFINED
           REF: 000014          ACON      4     +      STATIC         $MAIN

DATE
           DEF: 100218                                  CODE           DATE
           REF: 100EC0          ACON      4     +      CODE           DATE
                100EC4          ACON      4     +      CODE           DATE
                100EC8          ACON      4     +      CODE           DATE
                100ECC          ACON      4     +      CODE           DATE
                100ED0          ACON      4     +      CODE           DATE
```

```
VS LINKER ( 2.05.00)     12:11     11/01/88                XREF BY SYMBOL -  2

                                                                    SECTION
SECTION
SYMBOL     ADDRESS              TYPE     LEN    +/-     TYPE        NAME

           100ED4               ACON     4      +       CODE        DATE
           100ED8               ACON     4      +       CODE        DATE
           100EDC               ACON     4      +       CODE        DATE
           100EE0               ACON     4      +       CODE        DATE
           100F08               ACON     4      +       CODE        DATE
           000004               ACON     4      +       STATIC      $MAIN

MAIN
           DEF: 100044                                   CODE        #MAIN
           REF: 000000          ACON     4      +       STATIC      $MAIN

RINGBELL
           DEF: 10018C                                   CODE        #RINGBEL
           REF: 000024          ACON     4      +       STATIC      $MAIN
                000048          ACON     4      +       STATIC      $RINGBEL

SETALARM
           DEF: UNDEFINED
           REF: 00001C          ACON     4      +       STATIC      $MAIN

SETTIME
           DEF: UNDEFINED
           REF: 000020          ACON     4      +       STATIC      $MAIN

TIME
           DEF: 100170                                   CODE        TIME
           REF: 000008          ACON     4      +       STATIC      $MAIN

WLPALLOC
           DEF: 100F28                                   CODE        WLPALLOC
           REF: 00003C          ACON     4      +       STATIC      $MAIN

WLPFREE
           DEF: 100FB0                                   CODE        WLPFREE
           REF: 000040          ACON     4      +       STATIC      $MAIN
```

A.2.8 Program Link Map -- SSL Symbols and Assigned Aliases

```
VS LINKER ( 2.05.00)     12:11     11/01/88        SSL SYMBOLS AND ALIASES -  1

SSL SYMBOLS AND ASSIGNED ALIASES:

SSL SYMBOL                ASSIGNED ALIAS          INPUT FILE ALIAS COUNT

CRTCLOSE                  @CRTIO                          1
CRTOPEN                   @CRTIO                          1
CRTREAD                   @CRTIO                          1
CRTWRITE                  @CRTIO                          1
```

A.2.9 Program Link Map -- Undefined Symbols

```
VS LINKER ( 2.05.00)     12:11     11/01/88              UNDEFINED SYMBOLS -  1

UNDEFINED SYMBOLS:                      REFERENCING SECTIONS:

SETALARM
                                        STATIC   $MAIN
SETTIME
                                        STATIC   $MAIN
```

A.2.10 Program Link Map -- Output Statistics

```
VS LINKER ( 2.05.00)     12:11     11/01/88            OUTPUT FILE SUMMARY -  1

OUTPUT STATISTICS
    CODE SECTIONS LENGTH       0011D0 (     4560) BYTES
    STATIC SECTIONS LENGTH     0000E0 (      224) BYTES
    PROGRAM BASE ADDRESS       100000
    PROGRAM ENTRY POINT
                  ADDRESS      100044
                  NAME         MAIN
                  OFFSET       000000
    PROGRAM NAME
    RELEASE DATE               11/01/88
    VERSION NUMBER             00.00.00
    OBJECT FORMAT              1
    OUTPUT FILE                LANG.DRDRUN.CLOCK
    NUMBER OF RECORDS                8 RECORD(S)
```

A.2.11 Program Link Map -- Warning Messages

```
VS LINKER ( 2.05.00)     12:11     11/01/88              WARNING MESSAGES -  1

WARNING: There are unresolved external references.
```

A.3 EXAMPLE OF CREATING A SHARED SUBROUTINE LIBRARY (SSL)

This section gives an example of creating a shared subroutine library (SSL) from linking a set of subroutines. The subroutines in this example are assembled in a single object file called CRTSUBS, which is used as input to the Linker. Typically, subroutines are linked from multiple object files.

The entry point names selected for resolving external references are CRTOPEN, CRTCLOSE, CRTREAD, and CRTWRITE. The output file is called CRTSSL and is located in the system library @SYSLIB.

The system administrator uses the SSL utility to assign an alias when installing the files as a shared subroutine library. The SSL alias is the name that you use to specify the library containing shared subroutines that are called by your program. The alias for the shared subroutine library used in the previous example is @CRTIO. The subroutines in this example are, in fact, the shared subroutines called by the program in the previous example.

A.3.1 VS Procedure for Linking a Shared Subroutine Library (SSL)

```
PROCEDURE
RUN LINKER
    ENTER OPTIONS    SHAREDSL=YES,  ENTNAMES=YES
    ENTER INPUT      FILE=CRTSUBS,  LIBRARY=DRDOBJ,   VOLUME=LANG
    ENTER INPUT
    ENTER ENTRY      NAME1=CRTOPEN,
                     NAME2=CRTCLOSE,
                     NAME3=CRTREAD,
                     NAME4=CRTWRITE
    ENTER ENTRY
    ENTER OUTPUT     FILE=CRTSSL,   LIBRARY=@SYSLIB,  VOLUME=LANG
    ENTER PRINT      FILE=CRTSSL,   LIBRARY=@PRTLIB,  VOLUME=LANG,
        INPLOG       = YES,
        INPFILES     = YES,
        LINKSECT     = YES,
        DUPSECT      = NO,
        ADDRXREF     = YES,
        NAMEXREF     = YES,
        BASESECT     = YES,
        SSLSYMB      = NO,
        ENTNAMES     = YES,
        UNDEFINE     = NO,
        SUMMARY      = YES
```

A.3.2 SSL Link Map -- Input Log

```
VS LINKER ( 2.05.00)     12:11     11/01/88                    INPUT LOG -  1

INPUT LOG:

    OPTIONS:
        MAP                                 YES
        SYMBOLIC                            YES
        LINKAGE                             YES
        DATESEL                             NO
        DUPSECT                             NO
        REORDER                             NO
        RESOLVE                             NO
        OBJFORM                             1
        STATIC SUBROUTINE LIBRARY           NO
        SHARED SUBROUTINE LIBRARY           YES
        REVIEW ENTRY POINT NAMES            YES
        REVIEW ALIAS ASSIGNMENTS            NO

    INPUT FILE STRINGS:
        LANG  .DRDOBJ  .CRTSUBS

    SUBROUTINE LIBRARY STRINGS:
        NO SUBROUTINE LIBRARIES

    OUTPUT PARAMETERS:
        OUTPUT FILE                         LANG.@SYSLIB.CRTSSL
        FILE CLASS                          H
        RETAIN                                0 DAY(S)
        PROGRAM BASE ADDRESS                100000
        PROGRAM NAME
        ENTRY POINT NAME                    CRTOPEN
        ENTRY POINT OFFSET                  000000
        VERSION NUMBER                      00.00.00
        RELEASE DATE                        11/01/88

    PRINT LINK MAP:
        LINK MAP FILE                       LANG.@PRTLIB.CRTSSL
        INPUT LOG                           YES
        INPUT FILES                         YES
        LINKED SECTIONS                     YES
        DUPLICATE SECTIONS                  NO
        XREF BY ADDRESS                     YES
        XREF BY SYMBOL                      YES
        BASE-DEPENDENT SECTIONS             YES
        SSL SYMBOLS AND ALIASES             NO
        SUBROUTINE ENTRY NAMES              YES
        UNDEFINED SYMBOLS                   NO
        OUTPUT FILE SUMMARY                 YES
```

A.3.3 SSL Link Map -- Input Files

```
VS LINKER ( 2.05.00)     12:11     11/01/88                    INPUT FILES -  1

INPUT FILES:

FILE = LANG.DRDOBJ.CRTSUBS     SYMBOLIC = YES;  INPUT TYPE = USER

    SECTIONS INCLUDED:
        CRTIO
        $CRTIO
```

A.3.4 SSL Link Map -- Linked Code Sections

```
VS LINKER ( 2.05.00)     12:11     11/01/88                LINKED SECTIONS -  1

LINKED CODE SECTIONS:

ORIGIN LENGTH  SECTION NAME
        INPUT FILE                TRANSLATOR   VERSION   DATE      TIME

100040 000150  CRTIO
        LANG.DRDOBJ.CRTSUBS       AS           01.09.03  11/01/85  00:00
        *** ENTRY POINTS ***
        OFFSET  ADDRESS   NAME
        000000  <100040>  CRTIO
        000000  <100040>  CRTOPEN
        000024  <100064>  CRTCLOSE
        00004C  <10008C>  CRTREAD
        0000AC  <1000EC>  CRTWRITE
        0000E4  <100124>  CRTORDER
        00011C  <10015C>  CRTSTART
```

A.3.5 SSL Link Map -- Linked Static Sections

```
VS LINKER ( 2.05.00)     12:11     11/01/88                LINKED SECTIONS -  2

LINKED STATIC SECTIONS:

ORIGIN LENGTH  SECTION NAME
        INPUT FILE                TRANSLATOR   VERSION   DATE      TIME

000000 000090  $CRTIO
        LANG  .DRDOBJ  .CRTSUBS   AS           01.09.03  11/01/85  00:00
        *** ENTRY POINTS ***
        OFFSET  ADDRESS   NAME
        000000  <000000>  $CRTIO
```

A.3.6 SSL Link Map -- Cross-Reference by Address

```
VS LINKER ( 2.05.00)     12:11     11/01/88                XREF BY ADDRESS -  1

CROSS REFERENCE BY ADDRESS:

  ADDRESS   TYPE LEN +/-  REFERS TO SYMBOL     DEFINED AT   IN SECTION   TYPE

CODE   SECTION NAME = CRTIO
    100060  RCON  4   +   $CRTIO               000000       $CRTIO       STATIC
    100088  RCON  4   +   $CRTIO               000000       $CRTIO       STATIC
    1000E0  RCON  4   +   $CRTIO               000000       $CRTIO       STATIC
    100120  RCON  4   +   $CRTIO               000000       $CRTIO       STATIC
    100158  RCON  4   +   $CRTIO               000000       $CRTIO       STATIC
    100188  RCON  4   +   $CRTIO               000000       $CRTIO       STATIC

STATIC SECTION NAME = $CRTIO
    000000  ACON  4   +   CRTIO                100040       CRTIO        CODE
```

A.3.7 SSL Link Map -- Cross-Reference by Symbol

```
VS LINKER ( 2.05.00)     12:11     11/01/88                XREF BY SYMBOL -  1

CROSS REFERENCE BY SYMBOL:

SYMBOL    ADDRESS              TYPE     LEN   +/-   SECTION TYPE   SECTION NAME

$CRTIO
          DEF: 000000                                STATIC         $CRTIO
          REF: 100060          RCON     4     +     CODE           CRTIO
               100088          RCON     4     +     CODE           CRTIO
               1000E0          RCON     4     +     CODE           CRTIO
               100120          RCON     4     +     CODE           CRTIO
               100158          RCON     4     +     CODE           CRTIO
               100188          RCON     4     +     CODE           CRTIO
               000000          ACON     4     +     STATIC         $CRTIO

CRTCLOSE
          DEF: 100064                                CODE           CRTIO
          REF: NO REFERENCES

CRTIO
          DEF: 100040                                CODE           CRTIO
          REF: 000000          ACON     4     +     STATIC         $CRTIO

CRTOPEN
          DEF: 100040                                CODE           CRTIO
          REF: NO REFERENCES

CRTORDER
          DEF: 100124                                CODE           CRTIO
          REF: NO REFERENCES

CRTREAD
          DEF: 10008C                                CODE           CRTIO
          REF: NO REFERENCES

CRTSTART
          DEF: 10015C                                CODE           CRTIO
          REF: NO REFERENCES

CRTWRITE
          DEF: 1000EC                                CODE           CRTIO
          REF: NO REFERENCES
```

A.3.8 SSL Link Map -- Base-Dependent Code Sections

```
VS LINKER ( 2.05.00)     12:11     11/01/88        BASE DEPENDENT SECTIONS -  1

BASE DEPENDENT CODE SECTIONS:

THERE ARE NO BASE DEPENDENT CODE SECTIONS
```

A.3.9 SSL Link Map -- Subroutine Library Entry Point Names

```
VS LINKER ( 2.02.00)     12:11     11/01/88         SUBROUTINE ENTRY NAMES -  1

SUBROUTINE LIBRARY ENTRY POINT NAMES:

ADDRESS  ENTRY POINT NAME              SECTION NAME            TYPE

100064   CRTCLOSE                      CRTIO                   CODE
100040   CRTOPEN                       CRTIO                   CODE
10008C   CRTREAD                       CRTIO                   CODE
1000EC   CRTWRITE                      CRTIO                   CODE
```

A.3.10 SSL Link Map -- Output Statistics

```
VS LINKER ( 2.05.00)     12:11     11/01/88            OUTPUT FILE SUMMARY -  1

OUTPUT STATISTICS:
    CODE SECTIONS LENGTH:      000150 (      336) BYTES
    STATIC SECTIONS LENGTH:    000090 (      144) BYTES
    PROGRAM BASE ADDRESS       100000
    PROGRAM ENTRY POINT
                  ADDRESS      100040
                  NAME         CRTOPEN
                  OFFSET       000000
    PROGRAM NAME
    RELEASE DATE               11/01/88
    VERSION NUMBER             00.00.00
    OBJECT FORMAT              1
    OUTPUT FILE                LANG.@SYSLIB.CRTSSL
    NUMBER OF RECORDS                 2 RECORDS
```

APPENDIX B
DUPLICATE SECTIONS

B.1 INTRODUCTION

This appendix explains how duplicate sections occur and includes examples of how the Linker reports them. It also describes the default selection rules that the Linker uses to select a section when duplicate section names are encountered.

Duplicate sections are sections that have the same name, not necessarily the same content. When a duplicate code or static section is encountered during the input phase of the link, a choice must be made as to which one of the sections is to be included in the link. Other sections with the same name must be omitted because the linking process requires section names to be unique.

B.2 HOW DUPLICATE SECTIONS OCCUR

Duplicate code sections occur most frequently from linking modules that have common runtime routines previously linked in, either from a compilation phase or from an explicit link. Duplicate static sections can occur for the same reason and also from compilers such as PL/I that require external variables to be defined in every module that references them. Duplicate sections can also result when merging program modules that were developed independently.

Duplicate sections are usually not a concern unless there is a potential problem with conflicting versions of the same program modules or runtime routines, or if duplicate sections were entirely unexpected. Normally the Linker uses a set of default selection rules (see Table B-1) to automatically select which sections to include in the link. However, you can override the selection rules and make the choices yourself either through workstation interaction or through procedure specifications.

B.3 HOW DUPLICATE SECTIONS ARE REPORTED

When a duplicate section is omitted, the Linker flags the section name in the Linked Sections portion of the link map with the message, "Duplicate Section Was Omitted." This means that one or more other input files had a section with the same name and that these sections were excluded from the link. The name of the input file appearing above the message is the one from which the included section was selected. Section B.4 gives an example of this.

A detailed listing of duplicate sections can be obtained by specifying DUPSECT = YES on the PRINT screen (refer to Section 4.4.2). This portion of the link map lists all the duplicate section names alphabetically, and, for each section name, all the files in which they occurred. Other data for each section is also printed: the section type and length, the translator and version number, the date of translation, and whether or not symbolic data is included in the section. This information can be helpful in diagnosing problems resulting from duplicate sections and in resolving the conflicts. An example of the listing is shown in Section B.4.

B.4 PROGRAM LINK MAP -- LINKED STATIC SECTION WITH DUPLICATE SECTION OMITTED

```
VS LINKER ( 2.05.00)     17:55     11/05/88                LINKED SECTIONS -  2

LINKED STATIC SECTIONS:

ORIGIN LENGTH  SECTION NAME
       INPUT FILE                TRANSLATOR   VERSION   DATE      TIME

000000 000010  $MAINPRO
       LANG  .DRDOBJ  .MAINPROG  PL           02.01.00  11/05/88  00:00
       *** ENTRY POINTS ***
       OFFSET  ADDRESS   NAME
       000000  <000000>  $MAINPRO

000010 000008  $MOD1
       LANG  .DRDOBJ  .MOD1      PL           02.01.00  11/05/88  00:00
       *** ENTRY POINTS ***
       OFFSET  ADDRESS   NAME
       000000  <000010>  $MOD1

000018 000050  ACCOUNT
       LANG  .DRDOBJ  .MOD1      PL           02.01.00  11/05/88  00:00
       <<< DUPLICATE SECTION WAS OMITTED >>>                          <------
     *** ENTRY POINTS ***
       OFFSET  ADDRESS   NAME
       000000  <000018>  ACCOUNT

000068 000008  $MOD2
       LANG  .DRDOBJ  .MOD2      PL           02.01.00  11/05/88  00:00
       *** ENTRY POINTS ***
       OFFSET  ADDRESS   NAME
       000000  <000068>  $MOD2

000070 000058  $MOD3
       LANG  .DRDOBJ  .MOD3      PL           02.01.00  11/05/88  00:00
       *** ENTRY POINTS ***
       OFFSET  ADDRESS   NAME
       000000  <000070>  $MOD3
```

B.5 PROGRAM LINK MAP -- DUPLICATE SECTIONS

```
VS LINKER ( 2.05.00)    17:55 11/05/88                WARNING MESSAGES -  1
WARNING: Duplicate section names were encountered.
```

B.6 HOW THE LINKER RESOLVES DUPLICATE SECTION NAMES

This section describes how the Linker resolves duplicate section names. The selection rules that are used to resolve duplicate section names differ depending on whether you specified YES or NO in response to the DATESEL option on the MOREOPT screen.

If you specify YES in response to the DATESEL option, the Linker will resolve duplicate sections according to the date of compilation. To do this the Linker inspects all sections in all of the specified input files and libraries. If any duplicates are found, the Linker selects the most recent version, based on the rules that are listed in Table B-1.

Table B-1. Selection Rules for Duplicate Sections

Condition	Applicable Rule
Sections are mixed types (one code, one static)	Linker generates an error and prompts user to select the correct section.
All code sections	If DATESEL=YES, the Linker selects the most recently compiled section. If DATESEL=NO, the Linker selects the first section encountered.
All static sections	If DATESEL=YES, the Linker selects the most recently compiled section. If DATESEL=NO, the Linker selects the first section encountered. The length of the selected section is extended to the longest instance.
All blank common static sections with initialization records[a]	ERROR[b]
All blank common static, length not necessarily equal[a]	Selects longest instance
All labeled common static, length not equal[a]	If one is initialized, the Linker selects the initialized section. If both are initialized, the Linker selects the longest. If both are the same length, the Linker selects the most recently compiled section (if DATESEL=YES) or the first section encountered (if DATESEL= NO).

(continued)

Table B-1. Selection Rules for Duplicate Sections (continued)

Condition	Applicable Rule
All labeled common static, length equal[a]	Selects first initialized section. The length of the selected section is preserved.
None of the above conditions apply or DATESEL=NO	Selects first section encountered.

[a] Applies to FORTRAN 66 and FORTRAN 77.

[b] If errors occur when executing in the foreground, the Linker displays the duplicate section name and the file name containing each section. You can then choose the desired section or cancel processing.

If the Linker is executing in the background when an error occurs, the error message is printed on the printer, and the Linker terminates.

B.6.1 Complications Encountered With Duplicate Section Names

The occurrence of two or more sections with the same name can present any of the following complications:

- The sections are not of the same type. For example, one is a code section and the other is a static section.
- The sections are of different lengths. Languages have different rules for how these sections should be treated.
- Any of the following specific semantic rules of FORTRAN, when running FORTRAN 66 and FORTRAN 77 are not satisfied:
  - FORTRAN generates two types of static sections in addition to the regular static section type generated by other translators: common static sections and labeled common static sections.

- Blank common static sections that contain multiple definitions can represent data areas of different lengths. However, labeled common static sections that contain multiple definitions should represent data areas of the same length.

- A blank common static section cannot have any initial values.

- For multiple occurrences of a blank common static section, the longest section should be selected.

- For multiple occurrences of a labeled common static section, the first initialized section should be selected. The length of the selected section is preserved.

The default selection rules do not apply for the following conditions:

- All blank common static sections with initialization records

- Different types of sections

If duplicate sections occur under one of these conditions, and you are running the Linker in the foreground, the Linker displays the duplicate section names and prompts you to make the selection. If you are running the Linker as a background task, an error message is printed on the printer, and the Linker terminates.

B.6.2 Information Maintained by the Linker to Resolve Duplicate Section Names

The Linker collects the following information so that the default selection rules can resolve most of the possible complications:

- The Linker maintains a list of currently selected sections. When a new file is processed, and any of its sections are duplicated, the duplicates are compared for type and length against currently selected sections.

- The Linker distinguishes between the following types of sections as provided in the object code format:

  - Code section
  - Regular static section
  - Blank common static section
  - Labeled common static section

APPENDIX C
BASE-DEPENDENT CODE SECTIONS

C.1 INTRODUCTION

This appendix explains what base-dependent code is, why it cannot be included in a shared subroutine library (SSL), and what can be done when base-dependent code is reported.

C.2 WHAT BASE-DEPENDENT CODE IS

Base-dependent code is code that must be executed at a certain location in the user's address space in order to run correctly. In other words, it is code in which there are address references that cannot be changed (relocated) by the Loader at runtime. Base dependency results when address constants in code sections of a program refer to symbols defined in code sections of the same program module or in other program modules. The values of these address constants are computed by the Linker as it resolves references to the code sections relative to a base address. If these constants are located in unmodifiable code segments, the Loader cannot change their values at runtime. Therefore the code must be loaded and executed at the same base address.

Base-dependent code is encountered most frequently in modules that have been written in Assembly language, because Assembly language allows address constants anywhere in the program. While there is no harm in this if the code is not intended to be relocated at execution time, it is good coding practice to put address constants in static sections only. In this way subroutines can be either statically or dynamically linked in a program.

Some compilers do not support relocation at runtime; that is, they generate address constants in code sections. Programs compiled with these compilers cannot be used as shared subroutines.

C.3 WHY BASE-DEPENDENT CODE CANNOT BE INCLUDED IN A SHARED SUBROUTINE LIBRARY

Shared subroutines are mapped into the user's address space by the Loader when a program that calls them is brought into memory for execution. The location at which the subroutines are mapped is determined at runtime, not at linktime. Therefore, all external references that the shared subroutines make to code sections within the library have to be relocated according to the actual base address at which the subroutines are mapped. The references are made through address constants, and the values of these constants have to be changed.

Because code sections in shared subroutines are shared by all the programs that call them, the code cannot be modified in any way without affecting other programs. This applies to address constants in code sections as well. On the other hand, each program gets a private copy of the static sections mapped into its modifiable data area. This allows address constants in static sections to be changed to relocate external references.

When a shared subroutine library is created, the Linker assigns values to address constants based on static resolution of external references within the library. These addresses are meaningless, however, when the subroutines are dynamically linked to a program. If a program calls a base-dependent subroutine from a shared subroutine library, external references are not properly resolved and a program exception or other abnormal program behavior results.

C.4 WHAT TO DO WHEN BASE-DEPENDENT CODE IS REPORTED

If a subroutine is found to be base-dependent when attempting to include it in a shared subroutine library, there are two possible courses of action. One is to remove the subroutine from the library and place it in a static subroutine library instead. The other is to recode the subroutine, if possible, to remove the base dependencies.

The second course of action is possible only if the subroutine is written in Assembly language and the source code is available. If a base-dependent subroutine is written in a compiled language, then the compiler itself is responsible for the problem and nothing short of changing the compiler can correct it. In such a case, it is necessary to remove the subroutine from the SSL and put it in a static subroutine library.

To correct base-dependent code written in Assembly language, you must first locate the base dependencies. You can locate them in a source listing by examining each code section reported as base-dependent and finding all A-type and V-type address constants. These constants must be moved to static sections. In addition, any R-type address constant that references a symbol defined in a code section must also be moved to a static section. (Note that this is not a legitimate use of R-type address constants, but the Assembler does not prevent it.) R-type constants in code sections that reference symbols defined in static sections do not have to be moved.

If you have trouble locating all base-dependent address constants in the source listing, an address cross-reference produced by the Linker will aid you in finding them. To get the cross-reference, run the Linker again and select the option to print a cross-reference listing in address order (see Section 4.4.2). This listing shows all of the address constants contained in each section. An example of the cross-reference listing is shown in Section C.7.

In the cross-reference, address constants are identified as either A-type (ACON) or R-type (RCON). V-type address constants are also identified as ACON since they are actually A-type constants that reference external symbols. The list of address constants must be examined carefully for each *code* section reported as base-dependent. Any A-type address constants that you find, and any R-type address constants that reference symbols defined in code sections, must be moved to static sections by making the appropriate changes in the source code and reassembling the program.

An example of how the Linker reports base-dependent code is shown in the following two sections. A section of the link map called "Base-Dependent Code Sections" (Section C.5) lists the code sections found to be base dependent. This list is printed by default if the option to create a shared subroutine library is specified on the OPTIONS screen. The last section of the link map called "Warning Messages" (Section C.6) shows the warning message for base-dependent code.

In the following link map section, the code section named FILREAD is reported to be base-dependent. The cross-reference listing (Section C.7) shows that FILREAD does indeed have an A-type address constant (ACON). This constant must be moved to a static section to correct the base dependency.

C.5 SSL LINK MAP -- BASE-DEPENDENT CODE SECTIONS

```
VS LINKER ( 2.05.00)     12:19     10/21/88     BASE DEPENDENT SECTIONS -  1

BASE DEPENDENT CODE SECTIONS:

      FILREAD
```

C.6 SSL LINK MAP -- WARNING MESSAGES

```
VS LINKER ( 2.05.00)     12:19     10/21/88                WARNING MESSAGES -  1

WARNING: Code sections of this file are base dependent requiring execution at
         the indicated base address.  This file cannot be included in a shared
         subroutine library (SSL).
```

C.7 SSL LINK MAP -- CROSS-REFERENCE BY ADDRESS

```
VS LINKER  ( 2.05.00)     12:23     10/21/88                XREF BY ADDRESS -  1

CROSS REFERENCE BY ADDRESS:

    ADDRESS  TYPE LEN +/-  REFERS TO SYMBOL     DEFINED AT   IN SECTION   TYPE

CODE   SECTION NAME =  FILREAD
100064     ACON    4   +   <LOCAL REFERENCE>
100068     RCON    4   +   $FILREAD             000000       $FILREAD     STATIC

STATIC SECTION NAME =  $FILREAD
000000     ACON    4   +   <LOCAL REFERENCE>
00000C     ACON    4   +   <LOCAL REFERENCE>
000010     ACON    4   +   FILREAD              100040       FILREAD      CODE
000014     ACON    4   +   FILREAD              100040       FILREAD      CODE
000018     ACON    4   +   FILREAD              100040       FILREAD      CODE
```

APPENDIX D
ERROR MESSAGES

D.1 INTRODUCTION

This appendix lists the error messages for the Linker as well as the error codes returned by the Linker to the calling program. Each message includes a brief description.

D.2 LINKER ERROR MESSAGES

Cannot access named volume and library. Please respecify or try again.

1. Check the spelling of volume and library names.
2. Make sure that the volume and library exist.
3. Make sure that the volume is not mounted for exclusive use by another user.

Cannot verify aliases - nonexistent or inaccessible system alias file.

The locations of the shared subroutine libraries installed on your system are kept in a system file, managed by your system administrator. Either this file does not exist or the system administrator is updating the file while your program is trying to access it. Try again or notify your system administrator.

Duplicate input string. Please respecify.

You have already specified this input file. If there are no more input files, leave the input string blank, and press ENTER.

Duplicate subroutine library string. Please respecify.

You have specified a subroutine library string more than once. Specify another string. If there are no more subroutine libraries, press ENTER, leaving the last line of the screen blank.

File cannot be opened. Please respecify or try again.

The input file or subroutine library file that you specified cannot be opened. Check to be sure that you have the right file access, and that the file is not in use, or enter another file name.

File is in use as an input file. Alter REPLACE option or respecify file.

Change REPLACE option to YES or enter a different output file specification.

File is not a subroutine library file. Please respecify.

The specified file is not a qualified subroutine library file. Please respecify.

File not found. Please respecify.

Recheck your specification.

Incomplete input string specification.

Make sure that you have specified either of the following input specifications:

- A volume name and a library name for a partially qualified name
- A volume name, a library name, and a file name for a fully qualified name

Incomplete link map file specification.

The link map file specification must be a fully qualified name unless you enter '##' for the file name to create a temporary file. For a fully qualified name, enter a volume name in the first field, a library name in the second, and a file name in the third.

Incomplete output file specification.

The output file specification must be a fully qualified name unless you enter '##' for the file name to create a temporary file. For a fully qualified name, enter a volume name in the first field, a library name in the second, and a file name in the third.

Input file is invalid - no linkage data. Please respecify.

The specified file does not contain linkage data and cannot be used for linking. Please choose another file.

Input file is not a valid program file. Please respecify.

1. Make sure that the information entered is specified in the correct order, i.e., volume, library, file.
2. Check spelling.
3. Make sure that the file is a program file.

Invalid file class.

The valid file classes are blank, #, $, @, A-Z.

Invalid offset.

The offset must be entered in hexadecimal digits (0-9, A-F), and the range must be between 000000 and FFFFFF.

Invalid program entry name.

The program entry point name must be the name of an included section name or an entry point name defined in an included section.

Invalid program file base address.

The base address must be entered in hexadecimal digits (0-9, A-F), and the range must be between 000000 and FFFFFF.

Invalid subroutine library string. Please respecify.

The specified subroutine library string is not a valid entry.

1. Make sure that you have specified the volume name in the first field, and the library name in the second field for a partially qualified name; or the volume name in the first field, library name in the second field, and file name in the third field for a fully qualified name.

2. Check the spelling of the subroutine library information.

3. Make sure that the file, library, and volume exist.

Library not found. Please respecify.

The library specified does not exist. The volume or library name may be incorrect; enter the correct names.

Link map file already exists. Press PF3 to scratch the file, or respecify.

The print file you specified already exists. You can replace the existing file with the link map file by pressing PF3. The existing file will be destroyed.

If you want to retain the existing file, change the FILE information on the PRINT screen.

Link map file cannot be scratched. Please respecify.

You pressed PF3 to replace an existing link map file with the print file information entered on the PRINT screen. However, the file cannot be scratched at this time. Please enter a different FILE specification.

Linkage data must be retained in order to access symbolic data.

Symbolic data is useless without accompanying linkage data. Please adjust options accordingly.

Linkage data must be retained when creating static subroutine libraries.

If you are creating a static subroutine library, the linkage data must be retained in order to use the file in future links. Either respecify YES for the LINKAGE option or NO for the STATICSL option on the OPTIONS screen.

Linker Help text is not available.

Either the INFO utility or the Linker Help text is not correctly installed to display the Linker Help text. Notify your system administrator.

No program files can be selected within specified library. Please respecify.

For an input file to be selected, it must be the correct file type, and it cannot be included more than once.

Output file already exists. Press PF3 to scratch the file, or respecify.

The output file that you have named already exists. You can replace the existing file with the output file by pressing PF3. The existing file will be destroyed. If you want to retain the existing file, respecify the FILE information on the OUTPUT screen.

Output file cannot be scratched. Please respecify.

You pressed PF3 to replace an existing file with the output file; however, the existing file cannot be scratched at this time. Enter a different file specification.

Output file is a subroutine library file currently in use. Please respecify.

A subroutine library file currently in use cannot be specified as the output file. Please specify another name or location for the output file.

Please position cursor.

The cursor is not positioned in a field. Position the cursor at the start of a field.

Please specify whether creating a STATIC or a SHARED subroutine library.

You have entered YES for both the STATICSL and SHAREDSL options. Change one of the options to NO.

Section name or prefix not found.

The section name that you are searching for does not exist. Check to be sure it is correctly specified.

There is no SSL with the specified alias. Please respecify.

An alias was specified for which there is no corresponding shared subroutine library installed on this system. To assign this alias to undefined symbols, either the corresponding shared subroutine library must be installed or you must explicitly designate alias assignments through the SSLALIAS screen (Figure 4-18).

There must be at least one input file.

Control cannot pass the input stage until at least one valid input file has been accepted. Specify an input string for Linker processing to continue.

The SSL named by the specified alias exists but cannot be accessed.

The Linker was not able to access the SSL. Try again, or notify your system administrator.

Unable to link to the DISPLAY utility.

The Linker was unable to transfer control to the DISPLAY utility to display the link map. Check with your system administrator to make sure that the DISPLAY utility is installed.

Value not in list.

The value typed in the field is not valid. Check the spelling. If the spelling is correct, press the Help PF key for information on valid responses, or consult the reference manual.

D.3 LINKER ERROR CODES

One of the following error codes is returned to the calling program upon completion of processing:

Code 0 Successful completion of the link.

Code 4 A warning message. The Linker issues a warning message if it encountered and processed duplicate sections. If the link map option was set to YES, the warning message is printed at the end of the link map. If the link map option was set to NO, the warning message is placed in a warning message file.

Code 8 A fatal error message. The Linker issues a fatal error message if it detected unresolved symbols or if it encountered multiple entry points. Fatal error messages are also listed in the link map, or, if no link map is specified, in the message file.

Code 16 An abort operation error message. The Linker issues an abort operation error message if the user stopped the execution of the Linker.

APPENDIX E
GLOSSARY

Address

A 6-digit hexadecimal value that describes a program code or data location.

Alias

A logical name of 40 characters or less that corresponds to a shared subroutine library. The Linker uses aliases to associate SSL symbols with shared subroutine libraries at linktime. At runtime, when a program section refers to a module contained in a shared subroutine library, the Loader uses the alias of the shared subroutine library to link modules into the program.

Background Link

The process of running the Linker entirely by procedure with no user interaction. The procedure is submitted as a background task, and the system enters it on the procedure queue.

Default Selection Rules

The procedure that the Linker follows to automatically resolve duplicate section names.

Defined Symbol

A symbol that has been assigned to a particular address. This is due to its association with a section name or entry point name within a section, which is included in a linked program.

Designated Entry Point Name

An entry point name that can be referred to from outside the program file in which the symbol is defined. This entry point name is used for determining if the program file contains pertinent code for either shared or static subroutine library linking operations.

Duplicate Section Names

Two or more sections from different program files with the same section name. The name of each section included in a VS program file must be unique.

Dynamic Link

A link of shared subroutines from a shared subroutine library. A dynamic link occurs at runtime.

Entry Point

The point that initiates the processing of the linked program or module; that is, the point at which the execution of the program or module begins.

Entry Point Name

A named location in a program module that can be referred to from outside that module. The corresponding reference is an external reference.

Entry Point Reference (EPR) Table

Contains all subroutine library entry point names. The Linker builds an EPR table if STATICSL = YES or SHAREDSL = YES on the OPTIONS screen.

Exclude

The process of omitting one or more sections from the linking process.

Foreground Link

The process of running the Linker by procedure with possible user interaction or as a workstation task.

Fully Qualified Name

An entity's volume, library, and file names, all specified together.

Help Text

On-line reference information about the Linker displayed through the VS INFO utility. To access Help text for the Linker, press PF13.

Included Section

A section of either an input file or static subroutine library, which becomes part of the resultant output file.

Input File Specification

The specification that determines one or more program files. For a partially qualified input specification, the library and volume must be entered. For a fully qualified input specification, the file, library, and volume must be entered.

Input Files

Files specified by the user for input to the Linker, comprising code and static sections that are to be part of the resultant output file.

Input Log

A link map section that contains all the data and control input provided by the user.

Interactive Link

In an interactive link, the Linker displays each appropriate screen according to the Linker options chosen. You interact with the linking process by managing the options and entering required information on a screen-by-screen basis. The link is complete after you have managed all appropriate screens, provided the necessary data, and the Linker has built the output file.

Link Map

The report of a Linker run.

Linked Sections

The set of sections that make up the output file.

Output Program File

The program file produced by a Linker run.

Partially Qualified Name

An entity's volume and library names specified together, but not the file name.

Prname

The screen label needed in VS Procedure language to qualify references to the fields or PF keys of a particular screen interaction.

Procedure

A VS Procedure language program.

Procedure Interpreter

A system program that processes each line of a Procedure language file individually. The Procedure interpreter does not create a program file from the procedure file.

Program Entry Point

The location in a program at which execution begins.

Program File

A consecutive VS file of 1024-byte records which complies with a valid object code format, and has the program file attribute.

Resolution

The process of associating external references with a defined symbol or an alias.

Selection Rule

See Default Selection Rules.

Shared Subroutine Library

A program file that contains subroutines which are selectively linked at runtime (dynamically) as opposed to linktime, and can be shared by multiple programs that are running concurrently.

Shared Subroutine Library Symbol

A symbol referenced in a program that has an associated alias. Shared subroutine library symbols are assumed resolved by a designated entry point name within a shared subroutine library.

Static Link

A link at linktime.

Static Section

A section in the program file that contains information required by the operating system to build and initialize memory for the data portion of a program run. There are two special types of static sections, blank common and labeled common static sections, which meet special FORTRAN needs.

Static Subroutine Library

A library similar to a collection of input files, whose sections are only included if they resolve at least one undefined symbol. Program elements derived from a static subroutine library become a part of the resultant output file.

Subroutine Library

A collection of subroutines that can be selectively linked.

Symbol

The means to refer to a program location. In the Linker context, it may represent a section or an entry point name defined within a section. If a program section or entry point name exists for a symbol used in an external reference, the symbol is defined; if not, it is undefined.

Symbolic Reference

Use of a particular symbol in a place other than where it is defined. An internal symbolic reference refers to a symbol defined in the same section. An external symbolic reference refers to a symbol defined in another section.

INDEX

I

L

M

O

P

R

S

T

U

V

W

WANG

Customer Comment Form

Publication Number 715-1145

Title VS LINKER REFERENCE RELEASE 7 SERIES

Help Us Help You . . .

We've worked hard to make this document useful, readable, and technically accurate. Did we succeed? Only you can tell us! Your comments and suggestions will help us improve our technical communications. Please take a few minutes to let us know how you feel.

How did you receive this publication?

- ☐ Support or Sales Rep
- ☐ Wang Supplies Division
- ☐ From another user
- ☐ Enclosed with equipment
- ☐ Don't know
- ☐ Other _____

How did you use this Publication?

- ☐ Introduction to the subject
- ☐ Classroom text (student)
- ☐ Classroom text (teacher)
- ☐ Self-study text
- ☐ Aid to advanced knowledge
- ☐ Guide to operating instructions
- ☐ As a reference manual
- ☐ Other _____

Please rate the quality of this publication in each of the following areas.

	EXCELLENT	GOOD	FAIR	POOR	VERY POOR
Technical Accuracy — Does the system work the way the manual says it does?	☐	☐	☐	☐	☐
Readability — Is the manual easy to read and understand?	☐	☐	☐	☐	☐
Clarity — Are the instructions easy to follow?	☐	☐	☐	☐	☐
Examples — Were they helpful, realistic? Were there enough of them?	☐	☐	☐	☐	☐
Organization — Was it logical? Was it easy to find what you needed to know?	☐	☐	☐	☐	☐
Illustrations — Were they clear and useful?	☐	☐	☐	☐	☐
Physical Attractiveness — What did you think of the printing, binding, etc?	☐	☐	☐	☐	☐

Were there any terms or concepts that were not defined properly? ☐ Y ☐ N If so, what were they? _____

After reading this document do you feel that you will be able to operate the equipment/software? ☐ Yes ☐ No ☐ Yes, with practice

What errors or faults did you find in the manual? (Please include page numbers) _____

Do you have any other comments or suggestions? _____

Name _____ **Street** _____

Title _____ **City** _____

Dept/Mail Stop _____ **State/Country** _____

Company _____ **Zip Code** _____ **Telephone** _____

Thank you for your help.

Printed in U.S.A. 14-3140A 2-88

WANG

Cut along dotted line

Fold

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

WANG

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 16 LOWELL, MA

POSTAGE WILL BE PAID BY ADDRESSEE

Wang Laboratories, Inc.
Technical Publications Dept.
M/S 012-260
One Industrial Avenue
Lowell, Massachusetts 01851-9971

Fold

WANG

To Order by Phone, Call:
1-(800)225-0234
Telex 172108

Order Form for Wang Manuals and Documentation

① Customer Number (If Known)

② Bill To: Ship To:

③ Customer Contact: ( ) ( ) Phone Name

④ Date Purchase Order Number

⑤ Taxable Yes ☐ No ☐ ⑥ Tax Exempt Number

⑦ Credit This Order to A Wang Salesperson Please Complete Salesperson's Name Employee No. RDB No.

⑧ Document Number	Description	Quantity	⑨ Unit Price	Total Price

Sub Total	
Less Any Applicable Discount	
Sub Total	
Local State Tax	
Total Amount	

⑩ Authorized Signature Date

☐ Check this box if you would like a free copy of
WangDirect Software & Literature Catalog
(711-0888A)

Ordering Instructions

1. If you have purchased supplies from Wang before, and know your Customer Number, please write it here.
2. Provide appropriate Billing Address and Shipping Address.
3. Please provide a phone number and name, should it be necessary for WANG to contact you about your order.
4. Your purchase order number and date.
5. Show whether order is taxable or not.
6. If tax exempt, please provide your exemption number.
7. If you wish credit for this order to be given to a WANG salesperson, please complete.
8. Show part numbers, description and quantity for each product ordered.
9. *Pricing extensions and totaling can be completed at your option: Wang will refigure these prices and add freight on your invoice.*
10. Signature of authorized buyer and date.

Wang Terms and Conditions

1. **TAXES** — Prices are exclusive of all sales, use, and like taxes.
2. **DELIVERY** — Delivery will be F.O.B. Wang's plant. Customer will be billed for freight charges; and unless customer specifies otherwise, all shipments will go best way surface as determined by Wang. Wang shall not assume any liability in connection with the shipment nor shall the carrier be construed to be an agent of Wang. If the customer requests that Wang arrange for insurance the customer will be billed for the insurance charges.
3. **PAYMENT** — Terms are net 30 days from date of invoice. Unless otherwise stated by customer, partial shipments will generate partial invoices.
4. **PRICES** — The prices shown are subject to change without notice. Individual document prices may be found in the WangDirect Software & Literature Catalog (711-0888A)
5. **LIMITATION OF LIABILITY** — In no event shall Wang be liable for loss of data or for special, incidental or consequential damages in connection with or arising out of the use of or information contained in any manuals or documentation furnished hereunder.

Printed in U.S.A. 14-3141A 2-88

WANG

Cut along dotted line

Fold

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

WANG

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 16 LOWELL, MA

POSTAGE WILL BE PAID BY ADDRESSEE

WangDirect
Wang Laboratories, Inc.
M/S 017-110
800 Chelmsford Street
Lowell, Massachusetts 01851-9972

Fold

WANG

ONE INDUSTRIAL AVENUE, LOWELL, MA 01851
TEL. (508) 459-5000, TELEX 172108

Printed in U.S.A. 715-1145 10-88